МІНІСТЕРСТОВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп‘ютерної та програмної інженерії
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А.С. Савченко
«____» _____ 20___р

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СПУПЕНЯ «БАКАЛАВР»

Тема: «Тестування ПЗ фірми «YouIP»»

Виконавець: студентка УС-412 Коваленко Марія Вікторівна
(студент, група, прізвище, ім'я, по батькові)

Керівник: к. т. н., доцент Моденов Юрій Борисович
(науковій ступень, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер: ст. викл. Шевченко О.П.
(П.І.Б.) (підпис)

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп‘ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ
Завідувач кафедри

А.С. Савченко
“_____”_____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту

Коваленко Марії Вікторівни

(прізвище, ім’я, по батькові)

1. Тема дипломного проекту Тестування ПЗ фірми “YouIP” затверджена наказом ректора № 636/ст. від 22.04.2021р.
2. Термін виконання проекту: з “19” квітня 2021р. по “14” червня 2021 р.
3. Вихідні дані до роботи: тестування програмного забезпечення фірми “YouIP” на прикладі монітору М-01 да багатокористувацької панелі Р-01.
4. Зміст пояснювальної записки (перелік питань, що підлягають для тестування): вступ, огляд підходів до тестування, поняття багу та умови його існування, підготовка документації та звітів про виявлені дефекти, ручне тестування монітору на прикладі моделі М-01 та багатокористувацької панелі Р-01 і перевірка коректності інтеграцій між пристроями, висновки.
5. Перелік обов'язкового графічного (ілюстративного) матеріалу: загальний перелік огляду підходів до тестування, поняття багу та умови його існування, підготовка документації та звітів про виявлені дефекти, ручне тестування монітору на прикладі моделі М-01 та багатокористувацької панелі Р-01 і перевірка коректності інтеграцій між пристроями. Приклади написання тестової документації в хмарній системі управління.

КАЛЕНДАРНИЙ ПЛАН

	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Аналіз літератури та джерел за темою дипломного проекту.	11.05.2021р. – 12.05.2021р.	
2	Розробка та затвердження плану дипломного проекту.	13.05.2021р.	
3	Проведення консультації з науковим керівником щодо створення першого розділ.	14.05.2021р.	
4	Аналітичний огляд і постановка задачі.	15.05.2021р. – 18.05.2021р.	
5	Порівняльний аналіз існуючих видів тестування.	19.05.2021р. – 22.05.2021р.	
6	Підготовка документації, шаблону звіту баг-репорту, тестування монітору на прикладі М-01.	23.06.2021р. – 27.05.2021р.	
7	Тестування багатокористувацької панелі на прикладі Р-01 та інтеграції з М-01.	28.05.2021р. – 04.06.2021р.	
8	Висновки та оформлення пояснювальної записки дипломного проекту.	05.06.2021р. – 08.06.2021р.	
9	Підписання необхідних документів у встановленому порядку.	09.06.2021р. – 10.06.2021р.	
10	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту	11.06.2021р. – 14.06.2021р.	

Студент **(*Коваленко М.В.*)**

Керівник дипломної роботи (***Моденов Ю.Б.*** *)*

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Тестування ПЗ фірми «YouIP»» містить: 50 сторінок, 27 рисунків, 1 таблицю, 6 літературних джерел.

Об’єкт дослідження: програмне забезпечення на прикладі монітору M-01 та панелі P-01.

Предмет дослідження: ручне тестування програмного забезпечення фірми «YouIP».

Мета роботи: протестувати програмні продукти за допомогою мануального тестування, написати відповідну документацію.

Методи дослідження, технічні та програмні засоби: ручне тестування, порівняльний аналіз видів перевірки, обробка літературних джерел, робота із системою управління тестами.

Отримані результати та їх новизна: протестовано програмне забезпечення на прикладі монітору та багатокористувацької панелі. Продукт не адаптовано під потреби користувача, так як наразі присутні критичні дефекти. Отримано статистику про проведене тестування і його результати. Наразі, програма не готова до використання кінцевим користувачем.

РУЧНЕ ТЕСТУВАННЯ, ТЕСТОВА ДОКУМЕНТАЦІЇ, ТЕХНІКИ ТЕСТУВАННЯ ПЗ, ЗВІТИ ПРО ВИЯВЛЕНІ ДЕФЕКТИ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Android	Платформа та операційна система для мобільних телефонів, планшетів та комп'ютерів.
IEEE 829-2008 IEEE Standard for Software and System Test Documentation	Стандарт документації до тестуванню програмного забезпечення, що є важливим елементом та піднімає звичайні експериментальні дії до рівня тестування програмного забезпечення.
ASCII	Система кодів, у якій цифри, літери та знаки пунктуації поставлені у відповідність числам від 0 до 127 включно. Наприклад, 35 відповідає «#», а 66 — літері «B» у верхньому регістрі.
API	Прикладний програмний інтерфейс — набір чітко визначених методів для взаємодії різних компонентів та засобів для створення та тестування програмного забезпечення.
DTMF	Двотональний багаточастотний аналоговий сигнал, що використовується для набору телефонного номера.
Qase	Хмарна система управління тестуванням.
TMS	Системи управління тестами (Test Management System, TMS) використовуються для зберігання тестової документації та інформації про те, як довго проводилось тестування, які були визначені пріоритети відповідно до графіку та отримання інформації у звітах про етапи тестування та якість ПЗ.

GUI	Графічний інтерфейс користувача - тип інтерфейсу, що дозволяє користуватись пристроєм за допомогою графічних зображень та візуальних кнопок, на відміну від текстових інтерфейсів, заснованих на використанні, наборі та навігації по тексту.
Camdroid (Linux)	Одне з найбільш відомих прикладів розробки з відкритим доступом для користувача, вільне програмне забезпечення та загальна назва операційних систем на основі ядра UNIX.
SIP	Session Initiation Protocol (SIP) — перспективний і поширений стандарт протоколу передачі даних. Дозволяє передачу відео або звуку, віртуальну реальність, он-лайн ігри та миттєві повідомлення.

ВСТУП

Тестування програмного забезпечення - вивчає зв’язок між фактичною та очікуваною поведінкою програмного забезпечення у кінцевому результаті. У більш широкому сенсі тестування - це система контролю якості, яка включає в себе управління тестами, їх планування та впровадження, а також експериментальний аналіз.

Тестування в своїй основі широко використовується для двох цілей:

- Визначити випадки, коли поведінка програми є некоректною, неадекватною або несумісною з тестами.
- Показати розробникові та замовнику, що даний програмний продукт відповідає вимогам і має відповідний рівень якості.

Загалом, ключовим словом у тестуванні та розробці програмного забезпечення є якість, яка представлена стандартом ISO/IEC 25010:2011, що описує заявлені та безпосередні потреби різних зацікавлених сторін. В середньому модель якості продукту включає наступні характеристики:

- функціональність;
- сумісність;
- захищеність;
- надійність;
- портативність (мобільність);
- зручність використання;
- рівень продуктивності.

Структура та зміст документів, що супроводжують процес тестування, визначаються стандартом "IEEE 829-2008 IEEE Standard for Software and System Test Documentation". Але, на жаль, навіть неперервне, цілісне і ретельне тестування не може гарантувати відсутність помилок у програмному забезпеченні. Не всі помилки при розробці програмного забезпечення можливо виявити повністю, так як несправності бувають завжди і створити комплексний вхідний тест - задача

нездійсненна. Іншими словами, повністю перевірити програмне забезпечення неможливо: навіть перевірка найпростішої програми займе надто багато часу і більше не буде наповнене вигодою для виробництва. Але слід прагнути отримати 100% охоплення тестами функціоналу додатку, але ринкові умови часто вимагають скорочення часу розробки та тестування, щоб скоротити терміни і витрати на розробку.

Тестування присутнє на кожному етапі життєвого циклу ПЗ, починаючи від проектування на папері і закінчуючи невизначено довгим терміном експлуатації користувача. Тестування - це обов’язково поетапний, планомірний та систематизований процес. Це є дуже важливо, оскільки на розробку та тестування часто відводиться дуже мало часу, тому продуманий і систематичний підхід дозволяє виявити програмні дефекти швидше, ніж швидкий непослідовний тест.

РОЗДІЛ 1
КЛАСИФІКАЦІЯ ПІДХОДІВ ДО ТЕСТУВАННЯ ПЗ

1.1. Характеристика задачі тестування

Задача тестування – знайти розбіжності між очікуваним та фактичним результатом. Такі перевірки можна класифікувати за великою кількістю різних характеристик. Існує кілька критеріїв, для яких типи та класифікації тестів є загальними. Види перевірок сильно різняться залежно від завдання, що вони вирішують, а також від того, які технології були використані. Саме різниця у цілях завдання зазвичай призводить до необхідності використання різних видів тестів.

1.2. Огляд підходів до тестування

Створення правильно працюючої програми неможливо здійснити без її тестування. Якщо процес розроблено правильно, ПЗ матиме менше помилок і буде більш якісним. Існує безліч типів і методів, щоб перевірити, чи відповідає програмний продукт його вимогам. Зробивши детальний аналіз, тести поділено на функціональні та нефункціональні, розглянуто в чому їхні особливості та що найкраще найкращим чином слугуватиме для тестування програмного забезпечення (а саме, монітор M-01, панель P-01 та додаток Intercom) фірми «YouIP».

1. За запуском коду на виконання розділяють наступні типи:

- Динамічне тестування (Dynamic Testing) - тестування з запуском коду на виконання.

- Статичне тестування (Static Testing) - без запуску коду. В рамках цього підходу тестування можуть перевірятись документи, графічні прототипи, код додатків і т.д

2. За доступом до коду програми виділено три підтипи (рис. 1.1):

- Метод чорного ящика (Black Box Testing) - безпосереднього доступу до коду немає.

Немає значення, чи буде тестування функціональне або нефункціональне, воно не передбачає знання внутрішнього устрою компонента або системи. Тестована програма в даному випадку, як чорний непрозорий ящик, змісту якого тестувальник не бачить. Метою цієї техніки є пошук помилок в неправильно реалізованій або взагалі відсутній функції, помилки інтерфейсу та в структурах даних, їх організації доступу до зовнішніх баз даних, поведінки або недостатньої продуктивності системи.

Наприклад: При тестуванні панелі Р-01, монітору М-01 та додатку Intercom, не зважаючи на реалізацію, перевірено кнопки і поля введення, джерелом очікуваного результату виступатиме специфікація (вимоги до функціоналу на стадії проектування).

- Метод білого ящика (White Box Testing) – так можна класифікувати підтип даного набору, коли доступ є до всього коду.

Даний метод передбачає, що внутрішня реалізація системи вже відомі тестувальникові і чітко визначено, яким повинен бути результат після пройдених кроків перевірки. Знання всіх функцій програми, що тестується і її реалізації - обов'язкові для цієї техніки. Тестування білого ящика - поглиблення у внутрішній устрій системи, за межі її зовнішніх інтерфейсів.

Наприклад: При тестуванні панелі Р-01, монітору М-01 та додатку Intercom, спочатку було досконало ознайомлено із реалізацією коду для поля введення, визначено всі передбачені вводи користувача (як коректні, так і ні) та порівнено фактичний результат виконання програми з очікуваним. При цьому, очікуваний результат визначався саме тим, як у результаті мав спрацювати код програми.

- Метод сірого ящика (Grey Box Testing) – визначається тим, що доступ є

тільки до певної частини коду. Доступ може надаватись до декількох модулів програми, пов’язаних між собою спільним функціоналом або інтеграціями, або надаватись до певної частини.

Даний метод тестування програмного забезпечення передбачає комбінацію White Box і Black Box підходів. Тобто, внутрішній устрій програми нам відомо лише частково. Передбачається, наприклад, доступ до внутрішньої структури та алгоритмів роботи ПЗ для написання максимально ефективних тест-кейсів, але саме тестування проводиться за допомогою техніки чорного ящика, тобто, з позиції користувача. Цю техніку тестування також називають методом напівпрозорого ящика: щось ми бачимо, а що - ні.

Приклад: При тестуванні панелі Р-01, монітору М-01 та додатку Intercom, вивчимо код програми для того, щоб краще зрозуміти принципи її роботи і вивчити можливі шляхи її виконання. Ці знання допоможуть написати тест-кейс, який неодмінно буде перевіряти певну функціональність.

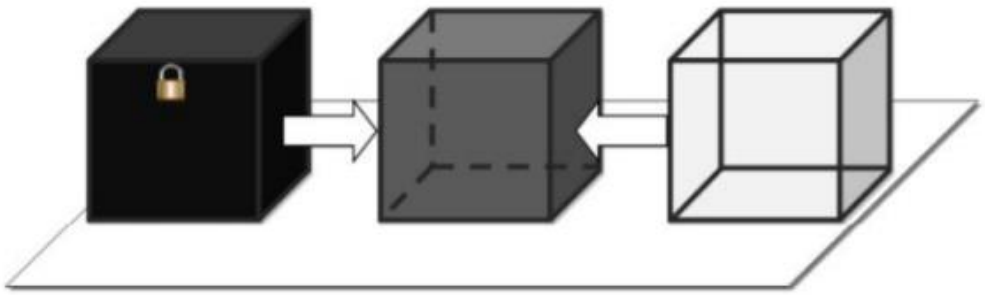


Рис. 1.1. Візуалізація виду тестування за доступністю до коду програми

3. Тестування за рівнем автоматизації:

- Ручне тестування - тестові кейси виконуються вручну людьми.

Тестувальники виконують перевірку ПЗ без використання будь-яких автоматичних інструментів. Це найнижчий і найлегший вид тестування, який не вимагає великих додаткових знань, але потребує логічного мислення та обізнаності у

користувацькому досвіді.

- Автоматизоване тестування - тестові кейси виконуються частково або повністю за допомогою спеціальних інструментів.

Цей підтип передбачає використання спеціального програмного забезпечення (не включаючи того, що тестується) для контролю виконання тестів та порівняння фактичних результатів програми. Даний метод допомагає автоматизувати ті тестові випадки, що необхідні для повного охоплення тестами функціоналу програми, але повторюються або на це витрачається велика кількість часу.

- Напівавтоматизоване тестування

Одним з основних принципів тестування є те, що 100% автоматизація неможлива. Тому ручне тестування є обов'язковим. Передбачається, що даний підтип буде використовуватися тільки для певних цілей, і ручний метод поєднуватиметься з прямим. Наприклад, з монітору М-01 можна подзвонити на багатокористувацьку панель Р-01 за допомогою автотесту, а відкрити двері через GUI вручну.

4. За рівнем деталізації ПЗ (за рівнем тестування):

- Модульне (компонентне) тестування (Module testing, Component testing) – тестуються окремі невеликі частини програмного забезпечення.

- Інтеграційне тестування (Integration Testing) - розглядає взаємодію між кількома частинами (модулями) програми.

- Системне тестування (System Testing) - програма, як правило, тестується як єдине ціле, всі компоненти мають бути охоплені.

5. За принципом роботи з програмним забезпеченням:

- Позитивне тестування - всі тестові сценарії з програмним забезпечення виконуються чітко згідно інструкції та без жодних недійсних дій, без введення неправильних даних і т.д.

- Негативне тестування – при роботі з програмним забезпеченням виконуються некоректні операції та використовуються дані, які можуть спричинити помилки (наприклад, введення значення у поле IP "0.0.0.0").

Негативні тести НЕ означають помилки в самій програмі. А навпаки, вважають,

що програма, яка працює вірно та стабільно, поводитиметься правильно навіть у найгірших ситуаціях (наприклад, з введенням значення у поле IP “0.0.0.0”, має відображатись повідомлення "Некоректне введення, перевірте правильність введених значень").

6. За рівнем підготовленості до тестування:

- Тестування по документації

Передбачає у собі перевірку правильності роботи додатку згідно з описаними вимогами (специфікацією).

- Інтуїтивне тестування (Ad-hoc)

Вид тестування, що проводиться без вимог та будь-яких документацій. Потреба у такому тестуванні виникає, коли тестувальники мають, наприклад, обмежений час і треба швидко дати рішення про готовність продукту, рівень його стабільності та працездатності, а можливість провести ретельне тестування за допомогою раніше підготовленого тест-плану, чеклисту, тест-кейсів відсутня.

7. За об’єктом тестування розділяють наступні підтипи:

- Функціональне тестування (functional testing) - описує функціональні можливості та особливості, а також взаємодії з іншими системами.

Такі тести можуть бути представлені на кожному рівні тестування: модульному (компонентному), інтеграційному та системному (див. п.4) Функціональне тестування вивчає зовнішню поведінку системи. Нижче наведено деякі найпоширеніші підтипи таких тестів:

- Тестування функціональності (Functional testing)

Вивчає заздалегідь описану поведінку додатку і базується на вимогах до очікуваного результату компонента або системи в цілому.

- Тестування безпеки та контролю доступу (Security and Access Control Testing)

Це стратегія тестування, яка використовується для перевірки безпеки системи, а також для аналізу ризиків, пов’язаних із забезпеченням цілісної системи захисту програмного забезпечення проти хакерів, вірусів та несанкціонованого

доступу до конфіденційних даних.

- Тестування взаємодій (Interoperability Testing)

Такі кейси перевіряють здатність програмного забезпечення взаємодіяти з одним або кількома компонентами, системами та включають у себе тестування сумісності та інтеграції.

- Нефункціональне тестування (Non-functional testing) - визначає тести, необхідні для визначення характеристик програмного забезпечення, яке може вимірюватись у певних величинах.

Іншими словами - це перевірка того, як система працює в цілому. Нижче наведено основні типи нефункціональних тестів. Даний підтип містить у собі усі види тестування продуктивності та всі види тестів, пов'язаних зі зімнами:

- Тестування продуктивності та навантаження (Performance and Load Testing)

Це автоматичне тестування, яке імітує роботу певної кількості бізнес-користувачів над певним (спільним) ресурсом.

- Стрес-тестування (Stress Testing)

Дозволяє оцінити, як система і система в цілому перебувають у стресовому стані, а також оцінити здатність системи до регенерації, тобто. повернутися до норми після припинення впливу.

- Тестування на стабільності або надійності (Stability / Reliability Testing)

Перевірка продуктивності програмного забезпечення під час тривалого за часом тестування, але середнього рівня навантаження.

- Об'ємне тестування (Volume Testing)

Збільшення об'єму у базі даних додатку для отримання оцінки продуктивності.

- Тестування інсталяції (Installation testing)

Даний підтип має на меті перевірити успішність інсталяції та налаштувань, а також оновлення та видалення програмного забезпечення.

- Тест на зручність використання (Usability Testing)

Цей метод тестування спрямований на встановлення рівня зручності розуміння, використання, здатності до навчання користування додатком та привабливості споживачеві товару.

- Тестування на збій та відновлення (Failover and Recovery Testing)

Перевіряє додаток на його здатність протистояти та відновлюватись після можливих збоїв через помилки самого програмного чи апаратного забезпечення, проблем у зв’язку (наприклад, збій мережі).

- Тестування конфігурацій (Configuration Testing)

Спеціальний тип тестування, який спрямований на моніторинг продуктивності програмного забезпечення в декількох конфігураціях системи (наприклад, на різних операційних системах, драйверах або з різним заповненням об’єму пам’яті тощо)

- Види тестування, пов’язані зі змінами:
  - Димне тестування (Smoke Testing, Build Verification Testing)

Даний вид охоплює тест-кейси на перевірку того функціоналу, що є найважливішим і яким користуються найчастіше, без якої використання програми буде марним. Також, так можна назвати першу перевірку програмного забезпечення (після написання коду або після внесення серйозних змін). Зазвичай його використовують для визначення готовності програми до подальшого, більш глибшого тестування.

- Санітарне тестування (Sanity Testing)

Вузьконаправлений вид тестування, результат якого достатній для того, щоб підтвердити, що певна функція працює згідно вимог.

Важливо: Деякі джерела помилково вважають, що санітарне та димне тестування - це одне і те ж. На мою думку, ці типи вимірювань мають "вектори руху" у абсолютно різних напрямках (рис. 1.2). На відміну від димного, санітарне тестування зосереджене на конкретній задачі, що тестується, тоді як димне розроблюється так, щоб охопити якомога більше результатів у найкоротші терміни.

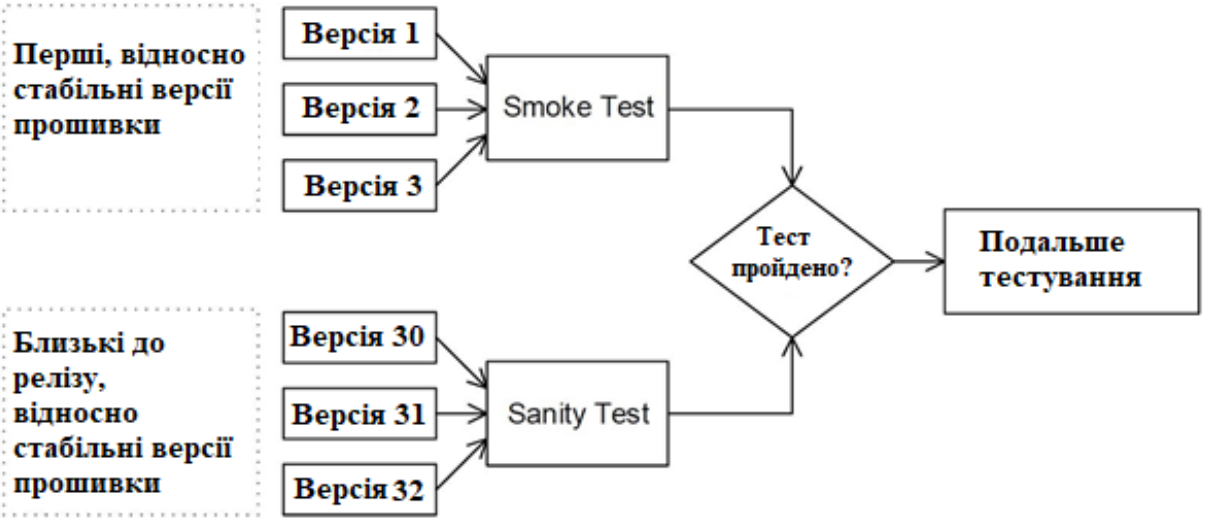


Рис. 1.2. Відмінності санітарного та димного тестування

- Регресійне тестування (Regression Testing)

Тестування, що має на меті підтвердити той факт, що попередні перевірки функціоналу не виявляли помилок, спричинених змінами в програмі чи її робочому середовищі. Регресія є важливим інструментом забезпечення якості і активно використовується у будь-якому проекті.

- Повторне тестування (Re-testing, Confirmation testing)

Проведення перевірок, які раніше виявляли помилки, щоб підтвердити той факт, несправність усунена. Насправді, цей тип тестування з'являється на останній стадії життєвого циклу багу, метою якого є перетворення статусу помилки з «потребує перевірки» на «закритий».

1.3. Техніки тест-дизайну

Методи (або прийоми) проектування тестів - це і є спосіб створення тестів. Ці методи мають теоретичний компонент (рекомендаційні інструкції із застосування), але основна їх частина тільки практичний. Це означає, що кожна техніка дає нам поради щодо того, як краще створити та описати той тестовий сценарій, що потрібен

саме вашій компанії, але як ми її використаємо залежить лише від тестувальника. Тому важливо не тільки навчитися техніці, а й практикувати її на практиці. Технічні працівники можуть надати вказівки не тільки щодо структури тесту (розробки тесту), а й щодо його впровадження.

Техніки тест-дизайну - це сукупність методів, які дозволяють створити невеликий набір даних/випадків, які максимально охоплюють функціонал, що знаходиться на тесті.

Розглянемо кілька основних прийомів проектування тестів:

- поділ на класи еквівалентності;
- аналіз граничних значень;
- пошук ортогональних масивів;
- причина-наслідок;
- прогнозування помилок.

Поділ на класи еквівалентності (Equivalence Class Partitioning) - мета цієї техніки зменшити кількість тестів, усунути їх надмірність і при цьому не втратити якість. Ця методика базується на аксіомі: якщо помилка виявляється при тестуванні в межах одного еквівалентного класу, то вона також буде виявлятися при тестуванні будь-якого іншого значення в цьому класі, і навпаки. Отже, ми розділили вхідні дані на діапазони і припускаємо, що тестування одного значення в рамках цього діапазону буде достатнім (рис. 1.3).

Рис. 1.3. Поділ на класи еквівалентності

Аналіз межових значень (Boundary Value Testing) – дана техніка допомагає

вибрати найбільш оптимальні та ефективний значення для тестування. Ця методика може застосовуватися до всіх рівнів тестування – unit-тестів, модульних, інтеграційних та системних.

Підхід до написання тест-кейсу визначається згідно таких критеріїв:

- Спочатку треба визначити діапазон значень (зазвичай, це раніше описаний клас еквівалентності).
- Потім визначити межі кожного діапазону. Саме ці межі надалі будуть виступати у ролі визначника граничного значення.
- Створити 3 тестові кейси для кожного обмеження - один, який перевіряє граничне значення, другий для значення нижче граничного, а третій для значення, що перевищує граничний.

Рис. 1.4. Аналіз межових значень

Тестування ортогональних масивів (Pair-wise Testing) - це сучасна та ефективна методика тестування, заснована на припущенні, що більшість помилок виникатиме тоді, коли не більше двох факторів впливатимуть один на одного.

Тестові набори, створені цим методом, охоплюють усі унікальні пари комбінацій, що вважається достатнім для виявлення додаткових дефектів. Принцип «pairwise» (з англ. pairwise - попарно) полягає в тому, що у переважній більшості випадків не потрібно проводити повномасштабний тест (тобто перевіряти усі ітерації у всіх конфігураціях, де значення всіх параметрів перетинають одне одного). А це часто неможливо через брак ресурсів (наприклад недостача часу або одиниць

штабу)[1].

Тому оголошено, що достатньо перевірити роботу програмного забезпечення, коли кожне значення кожного параметра зустрічається принаймні один раз. Бувають випадки, коли вже самі параметри є ортогональними і нам не потрібно їх «перетинати». З цієї причини тестові конфігурації зменшуються ще більше.

Наприклад, є дві операційні системи додатку Intercom: Android і Camdroid (Linux). Візьмемо три найбільш вживані діагоналі смартфонів для того, щоб перевірити графічний інтерфейс користувача: 5,5″, 4,7″ і 5,7″. В деяких випадках, через помилки у адаптації додатків під різні екрани, навіть міліметр різниці у діагоналі може суттєво змінити зовнішній вигляд сторінки додатку, що значно погіршить користувацький досвід.

Є 6 комбінацій (дві ОС помножені на три види діагоналей). Наш додаток підтримується двома мовами – українською та російською (UA та EN). Для повного покриття GUI тестами, необхідно помножити 6 на 2, тобто перевірити кожну з попередніх конфігурацій зі всіма підтримуваними мовами додатку Intercom.

Перенесемо описані послідовності дій до таблиці, щоб краще зрозуміти, як саме будуть між собою пов’язані дані, чи не упущено важливих деталей, а також чи є сенс у всіх перевірках чи варто скоротити обсяг роботи, прри цьому не втративши якість (табл. 1.1).

Таблиця 1.1

Тестування за технікою ортогональних масивів на прикладі додатку Intercom

№ кейсу	Діагональ	Операційна система	Мова
1	5,5″	Android	UA
2	5,5″	Android	EN
3	5,5″	Camdroid	UA

Продовження табл. 1.1

4	5,5″	Camdroid	EN
5	4,7″	Android	UA
6	4,7″	Android	EN
7	4,7″	Camdroid	UA
8	4,7″	Camdroid	EN
9	5,7″	Android	UA
10	5,7″	Android	EN
11	5,7″	Camdroid	UA
12	5,7″	Camdroid	EN

Після створення таблиці виникає питання: чи є сенс робити таку кількість перевірок? Адже можемо використати техніку «pairwise» і замість 12 конфігурацій отримати 6. Цей метод дає змогу визначитись, чи важливо нам перевіряти кожну ОС у співпраці з іншими мовами. На мою думку - ні, більш доречним було б приділити увагу тому, як працює операційна система з кожною мовою. Умови задачі виконуються - ми плануємо покрити тестами всі діагоналі на всіх платформах; кожну мову сайту зі всіма діагоналями та на кожній платформі. Звичайно, ідеальним варіантом було б, коли кожен параметр перетнеться з іншим лише один раз, але це можливо лише якщо у всіх параметрах однакова кількість значень, що рівна кількості параметрів.

Причина - наслідок (Cause/Effect – CE) - це спосіб розробки тестових кейсів, які забезпечують формальний запис логічних умов та пов'язаних з ними дій. Причиною в цьому випадку називають окремий клас вхідних умов або еквівалентностей. Результатом є вихідна умова або перетворення системи.

Наприклад, для панелі P-01 перевіримо можливість додавання клієнта за допомогою певної форми. Для цього вам потрібно буде перейти в налаштування пристрою, ввести значення у кілька полів, таких як «Будинок», «Парадне», «Індекс», а потім натиснути кнопку «#» - це є «Причина» (рис. 1.5). Після натискання на кнопки

«#» система додає клієнта до бази даних і відображає його логічну адресу при дзвінку та в меню викликів на моніторі M-01 - це "Наслідок"(рис. 1.6). Цей метод дозволяє побудувати дуже ефективні тести та виявити неповноту та неоднозначність початкових вимог.

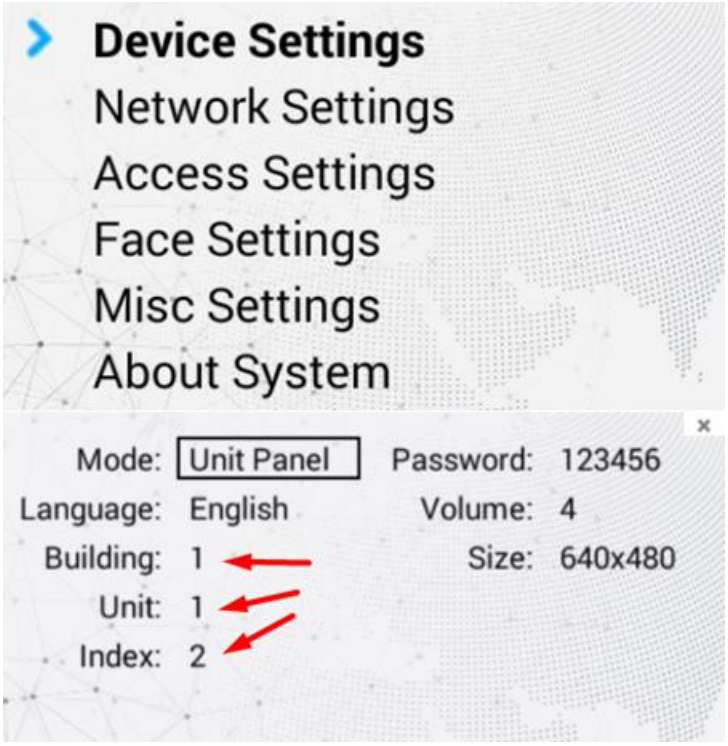


Рис. 1.5. Причина - налаштування монітору P-01

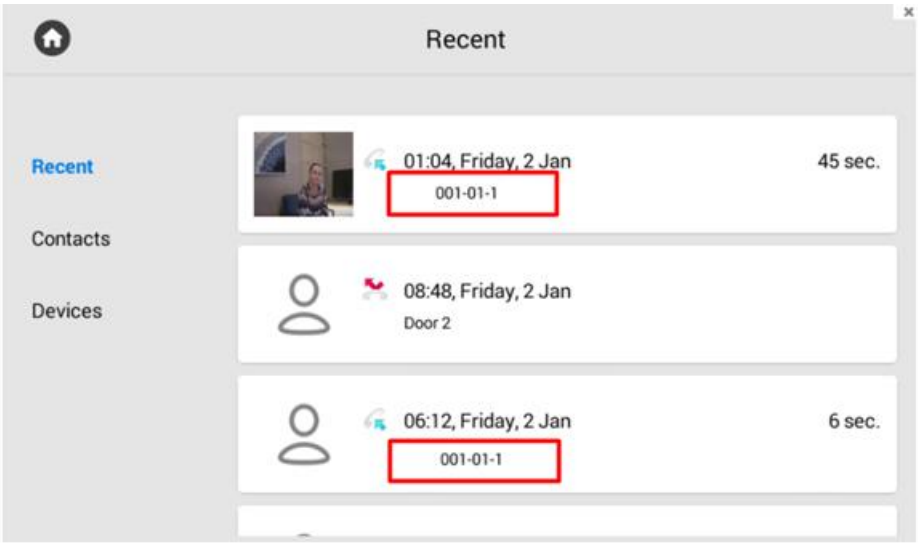


Рис. 1.6. Наслідок – вигляд логічної адреси на моніторі M-01

Передбачення помилки (Error Guessing - EG) - ця методика базується на аналітику, яка використовує свої знання про систему та її здатність інтерпретувати вимоги, щоб "передбачити", за яких вхідних умов система може генерувати помилку. Наприклад, у специфікації зазначено, що на головному екрані має бути показано повідомлення "Введіть пінкод", цифрова клавіатура та саме поле вводу. Я, як аналітик тесту думаю: "Що робити, якщо я не вводжу код?", "Що робити, якщо я вводжу неправильний код?" тощо. Це передбачає помилку. Маючи досвід, тестувальник розуміє типові вузькі місця, що дозволять ефективно прогнозувати помилки.

Рис. 1.7. Передбачення помилки – зовнішній вигляд екрану вводу пінкоду на панелі P-01

1.4. Поняття багу та умови його існування

Дефект (Defect, Bug) - відхилення фактичного результату (actual result) від очікувань користувача (expected result), сформованих на основі вимог, специфікацій, іншої документації або досвіду і логіки.

Очікуваний результат - поведінка системи, що описана в вимогах.

Фактичний результат - поведінка системи, що спостерігається в процесі тестування.

Дефекти бувають різних класифікацій в залежності від виду тестування. Наприклад, функціональне тестування, тестування документації, тестування

продуктивності і т.д. Звідси логічно випливає, що дефекти можуть зустрічатися не тільки в коді програми, а й в будь-якій документації, в архітектурі ПЗ, дизайні, в налаштуваннях тестованої програми або навіть оточення. Синонімами можуть виступати такі слова: помилка, відхилення, недолік, перешкода, відмова, проблема[2].

Назви етапів життєвих дефектів можуть бути різними в різних баг-трекінгових системах, але суть їх завжди лишається одною, тому прийнято виділяти такі стадії:

- Новий (New) - тестувальник знайшов баг, дефект успішно занесений в систему відстеження помилок.
- Відкрито (Opened) -після того, як тестувальник знайшов помилку, вона або автоматично, або вручну призначається на людину яка повинна її проаналізувати.
- Відкладений (Deferred) - виправлення цього бага не несе цінності на даному етапі розробки або по іншим причинам, щось відстрочує його виправлення.
- Відхилено (Rejected) - з різних причин дефект може і не рахуватися дефектом або вважатися неактуальним, що змушує відхилити його.
- Дублікат (Duplicate) - якщо описана помилка вже раніше була внесена в «Bug-tracking» систему, то статус такої помилки змінюється на «дублікат».
- Призначено (Assigned) - якщо помилка актуальна і повинна бути виправлена в наступній версії прошивки, відбувається призначення на розробника, який повинен виправити помилку.

Коли помилка виправлена розробником, статус може тільки значення:

- Виправлено (Fixed) – тобто відповідальний за виправлення цього дефекту розробник заявляє, що усунув дефект.

Залежно від того, чи виправив розробник дефект, статус може бути:

- Перевірено (Verified) – у цьому випадку тестувальник вже перевірив, чи дійсно відповідальний розробник виправив помилку і результат був позитивний.

Якщо баг виправлений і тестувальник перевірив, що дефекту дійсно немає, він отримує такі статуси:

- Повторно відкритий (Reopened). Якщо побоювання тестувальника виправдані і баги в новій версії ще присутні.

- Закритий (Closed). В результаті певної кількості циклів, баг все-таки остаточно усунутий і більше не потребує уваги команди – тоді він оголошується закритим.

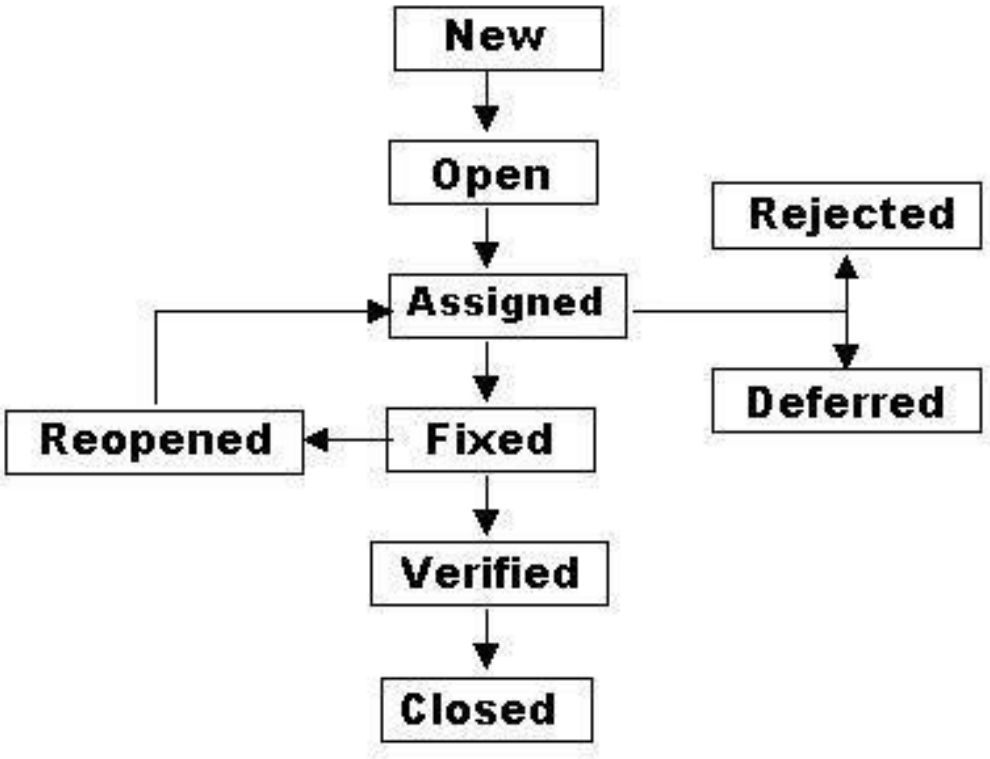


Рис. 1.8. Життєвий цикл багу

Класифікація дефектів, з точки зору ступеня впливу (Severity) на працездатність ПО:

- Blocker. Помилка, яка призводить програму в неробочий стан. Подальша робота з програмною системою або її функціями неможлива.
- Critical. Критичний дефект, що приводить деякий ключовий функціонал в неробочий стан. Також це може бути суттєве відхилення від бізнес логіки, неправильна реалізація необхідних функцій, втрата призначених для користувача даних і т.д.
- Major. Дуже серйозна помилка, яка свідчить про відхилення від бізнес

логіки або порушує роботу програми. Не має критичного впливу на додаток.

- Minor. Незначний дефект, що не порушує функціонал програми, але є невідповідністю очікуваному результату. Наприклад, помилка дизайну.
- Trivial. Баг, який не має вплив на функціонал або роботу програми, але який може бути виявлений візуально. Наприклад, помилка в тексті.

Градація дефектів, з точки зору пріоритетності виправлення (Priority):

- Високий (High) - баг повинен бути виправлений якомога швидше, тому що він критично впливає на працездатність програми.
- Середній (Medium) - дефект повинен бути обов'язково виправлений, але він не робить критичний вплив на роботу програми.
- Низький (Low) - помилка повинна бути виправлена, але вона не має критичного впливу на програму і усунення може бути відкладено, в залежності від наявності інших, більш пріоритетних дефектів.

1.5. Висновки до розділу

У ході написання даного розділу було проведено аналіз існуючих видів та підходів до тестування ПЗ. Було комплексно та повноцінно описано критерії, на які поділяються види тестування за їх суттю. Досліджено техніки тест-дизайну програмного продукту. На прикладі монітору М-01, панелі Р-01 та додатку Intercom було практично досліджено відповідні методи тестування, зроблено аналіз позитивних та негативних сторін кожного методу. Описано поняття багу та його життєвий цикл.

РОЗДІЛ 2
ТЕСТОВА ДОКУМЕНТАЦІЯ ТА ЗВІТИ ПРО ВИЯВЛЕНІ ДЕФЕКТИ

2.1. Тестування документації та вимог

Вимоги - опис того, як має себе поводити додаток в процесі вирішення корисної для користувача задачі.

У більш широкому розумінні можна сказати, що вимоги (англ. Requirements) - це сукупність тверджень щодо атрибутів, властивостей або якостей програмної системи, що підлягає реалізації.

На основі визначених умов до проекту складається наступний процес: планування, проектування, розробка та тестування. Описи очікуваних результатів роботи програми є основою для створення успішного продукту. Якщо ж спочатку вимоги не відповідають меті, не визначені або не узгоджені між учасниками проекту, то навіть найбільш скоординована робота команди проекту не призведе до успіху.

Якщо звернутися до визначення вимог у літературі 10-30 років тому, можна помітити, що спочатку взагалі не було згадок про користувачів, вирішення їх задач та корисних властивостей програми для них. Тобто користувач виступав, як абстрактні дані, не пов’язані з додатком. В даний час такий підхід є неприйнятним, оскільки він не тільки призводить до комерційного провалу товару на ринку, але й примножує витрати на розробку та тестування[2]. В ході аналізу, розглянуто основні рівні та типи вимог до функціоналу програмного забезпечення.

Бізнес-вимоги - визначають мету програмного забезпечення, описуються в документі про бачення, сферу дії та межі проекту.

Вимоги користувача - визначають набір завдань користувача, які програма повинна вирішувати, а також шляхи (сценарії) їх вирішення в системі. Запити користувачів можуть бути виражені у формі фраз ствердження, сценаріїв використання, історій користувачів та прикладів взаємодії.

Функціональні вимоги - включають передбачувану поведінку системи шляхом визначення видів діяльності, які система здатна виконувати. Описується в системній специфікації.

Нефункціональні вимоги - описують особливості системи (зручність використання, безпеку, надійність, масштабованість, продуктивність тощо), які повинно мати ПЗ при реалізації своєї поведінки.

Також, варто сказати про те, що кожні вимоги мають свої вимоги. Якісні характеристики по-різному визначаються різними джерелами. Однак загальновизнаними є такі характеристики:

1. Одиничність функції – вимога має описувати одну і тільки одну функцію.
2. Завершеність – характеристика повністю визначена в одному місці та містить всю необхідну інформацію.
3. Послідовність – опис функціоналу не суперечить іншим вимогам і повністю відповідає зовнішній документації.
4. Атомарність – вимога розглядається, як "атом". Це означає, що її неможливо розділити на кілька більш детальних вимог, не втрачаючи повноти загального опису.
5. Відстеження – характеристика повністю або частково відповідає потребам бізнесу, що раніше було зазначено зацікавленими сторонами та задокументовано.
6. Актуальність – описує те, що вимога з часом не застаріла і актуальна наразі.
7. Здійсненність – описана функціональність може бути виконана в рамках проекту.

8. Однозначність – вимога коротко визначена без використання технічного жаргону, абревіатур чи інших неявних мов і виражає об’єктивні факти, а не суб’єктивні думки. Можливе тільки одне тлумачення. Визначення не містить нечітких фраз. Забороняється використання негативних та складених претензій.

9. Обов'язковість – це характеристика, визначена зацікавленою стороною, відсутність якої призведе до неповноцінності рішення, ігнорувати дану умову ні в якому разі не можна. Необов’язкова вимога суперечить самому поняттю вимоги.

10. Верифікованість – впровадження вимоги можна визначити, використовуючи один із чотирьох можливих методів: огляд, демонстрація, випробування або аналіз.

Документація та випробування вимог класифікується, як нефункціональне тестування і було виділено такі основні методи:

- взаємний огляд;
- питання;
- тест-кейси та чек-листи;
- малюнки.

Взаємний огляд є одним із найбільш використовуваних методів для випробування вимог і може бути представлений в одній з трьох форм (по мірі зростання складності та збільшення витрат):

- Швидкий огляд – цю техніку можна порівняти із тим, коли ви та ваші одногрупники в університеті переглядали есе один одного, щоб з’ясувати, чи є помилки або дефекти у написанні тексту.
- Технічний огляд – цей вид можна уподібнити ситуації, коли угода затверджується юридичним відділом, бухгалтерією тощо.
- Формальний огляд – цей метод можна ототожнити загальному стану генерального прибирання квартири, коли перебирається вміст усіх шаф, холодильника, комори тощо.

Питання. Очевидною методикою тестування та поліпшення якості вимог є використання методів ідентифікації вимог та (як окремий вид діяльності) - задавання

питань. Якщо у вимогах хоча б щось викликає у вас нерозуміння або підозру - запитуйте. Для того, щоб протестувати систему згідно цього методу, можливо попросити представника замовника або звернутись до довідкової інформації. З багатьох питань, можливо і треба зв'язатися з більш досвідченими колегами, керівником, якщо вони мають будь-яку відповідну інформацію, що коли-небудь була отримана від замовника. Головне, щоб запитання були сформульовані так, що отримані відповіді могли тільки покращили вимоги.

Тестові кейси та чек-листи. Слід пам'ятати, що хороший запит буде перевірений, а це означає, що повинен бути об'єктивний спосіб з'ясувати, чи правильно виконано запит. Продумування чек-листыв або навіть детальні тестові випадки в процесі аналізу вимог дозволяють нам судити, як саме вимоги можуть бути перевірені. Якщо можливо швидко придумати декілька пунктів із чек-листу, це не означає, що запит відповідає вимогам (наприклад, він може суперечити іншим запитам).

Зображення (графічне зображення). Дуже зручно використовувати креслення, схеми, діаграми, розумові карти (англ. mind map) для того, щоб побачити загальний огляд вимог. Графічна презентація також підходить для наочності та висновків (наприклад, діаграма бази даних UML, яка займає один екран, може бути описана кількома десятками сторінок тексту). На малюнку дуже легко помітити, що деякі елементи не можуть вміститись, можуть бути відсутні і т.д. Якщо використовувати загальноприйняту нотацію для опису вимог (наприклад, UML, про яку вже згадувалося), можна отримати додаткові переваги: колеги легко зрозуміють і допрацюють ваші діаграми, а в підсумку може вийти гарне доповнення до текстовій формі уявлення вимог[3].

2.2. Документація: тест-план, чек-листи, тест-кейси та тест-сьюти

Під час тестування створювалась і використовувалась певна кількість тестових артефактів (документів, моделей тощо). Найпоширенішими артефактами тесту є: тестовий план(Test Plan), стратегія тесту (Test Strategy), чек-лист (Test Strategy), тест-кейс (Test Suite), набір тестів (Check List), звіт про тестування(Test Report)[1].

План тестування (Тест-план) - це документ, який описує конкретні знання та оцінки ризику з варіантами їх усунення, повний обсяг перевірок, починаючи від опису тестуємих об’єктів, стратегії, графіку, критеріїв початку та закінчення випробувань обладнання, необхідного в процесі експлуатації.

Документ повинен відповісти принаймні на такі запитання:

- Що треба тестувати? Тобто сам тестовий об'єкт: система, додаток, апаратне забезпечення тощо.
- Що буде тестуватись? Має бути наявний перелік функцій та компонентів системи, що перевіряється.
- Як буде тестуватись? Стратегія тестування має описувати типи тестування та їх використання відповідно до об'єкта тестування.
- Яким буде тестове оточення? Треба перевірити середовище, в яких повинен тестуватися програмний продукт. Це важливо для того, щоб тести були дійсними в умовах використання ПЗ користувачем.
- Коли буде проводитись тестування? Це означає, що треба описати послідовність робіт, а саме: підготовку, саме тестування, аналіз результатів.

Якщо ваша компанія має офіційно налагоджений процес, описати в плані тестування також буде доречним:

- список координаторів;
- прийняті стандарти та шаблони;
- критерії початку тестування:

- готовність випробувальної платформи (випробувальні стенди, робоче місце, яке може умістити усі пристрої і буде зручним у використанні для тестувальника);
- закінчення розробки необхідного функціоналу;
- наявність усієї необхідної документації;

- критерії завершення випробувань:
  - результати випробувань відповідають критеріям якості продукції;
  - дотримані вимоги щодо кількості не виправлених помилок;
  - конкретний період без зміни вихідного коду програми;
  - протягом певного періоду нові дефекти не знайдені.

Різні цілі та завдання тестування протягом життєвого циклу продукту призводять до необхідності розробки та впровадження різних стратегій тестування.

Тестова стратегія - це план тестування системи або її модуля з урахуванням особливостей функціональності та залежності від інших компонентів системи та платформи. Стратегії тестування необхідно розробляти на етапі планування тестування. Кожна така стратегія визначає:

- ітерації, для яких використовують стратегію тестування та цілі тесту;
- етапи тестування для кожної ітерації;
- критерії успішно проведеного тестування;
- типи тестів, що будуть використовуватись;
- набір методів та інструментів, необхідних для тестування та оцінки якості;
- критерії оцінки тесту.

Чек-лист є одним з основних інструментів тестування. Він дозволяє запам’ятовувати важливі тести, записувати результати вашої роботи та відстежувати статистику стану програмного продукту. Іноді контрольний список називають детальними інструкціями з випробування продукту, який містить послідовність дій, передумови, постумови та опис деталей. Але головний принцип чек-листу полягає в тому, що кожен тестувальник читає їх по-своєму і розширює набір тестів своїм

досвідом. Якщо розглядати переваги контрольних списків перед тестовими кейсами, можна виділити такі пункти:

- розширення охоплення тестами через різницю при проходженні;
- зменшення витрат на технічне обслуговування та підтримку тестування, так як не потрібно писати багато тексту, достатньо коротко написати послідовність дій та очікуваний результат;
- відсутність рутини, яка не подобається кваліфікованим тестувальникам і займає багато часу, який можна було б використати набагато продуктивніше;
- можливість виконувати та комбінувати тести по-різному, залежно від уподобань співробітників[4].

У той же час чек-листи мають ряд переваг, через які так популярні деталізовані тест-кейси (рис. 2.1):

- статистика: ким, коли, та які пункти були пройдені (із деталями версії прошивки продукту та середовища, в якому проводилося тестування);
- примітка, яка допоможе запам’ятати важливі тести;
- здатність оцінювати стан ПЗ, його готовність до випуску.

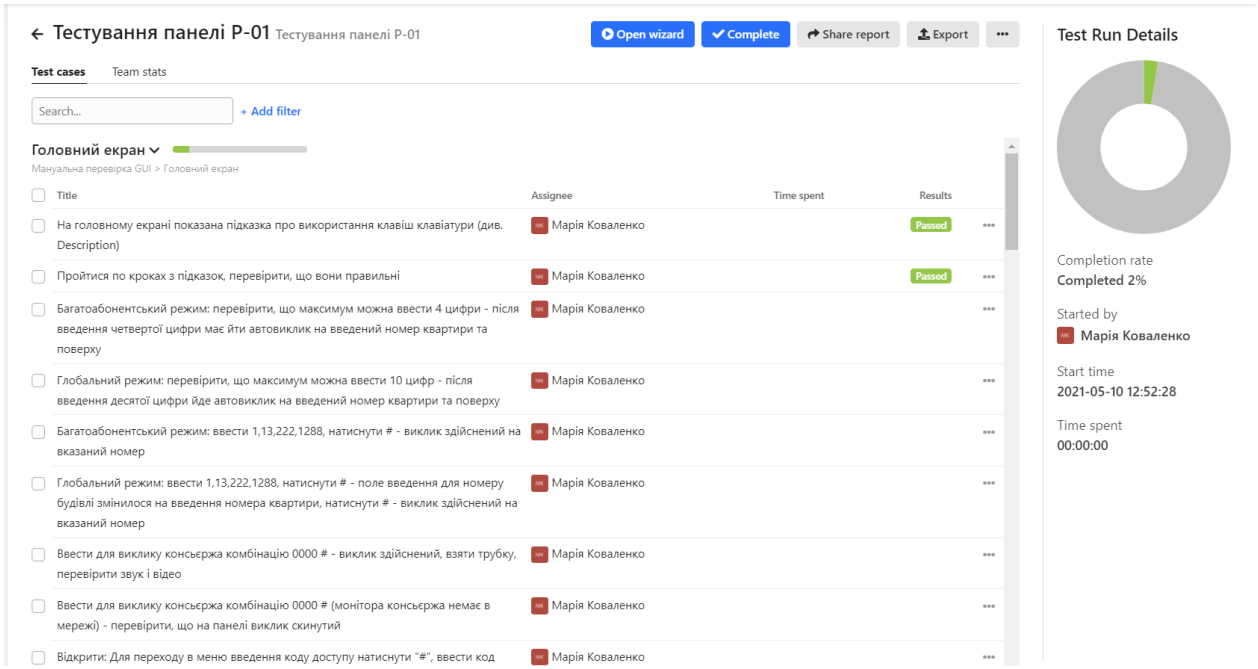


Рис. 2.1. Вигляд чек-листу на прикладі багатокористувацької панелі P-01

Ще однією обов'язковою сутністю, з якою була проведена робота, є тестовий випадок. Тест-кейс - це артефакт, суть якого полягає у виконанні певної кількості дій і/або умов, необхідних для перевірки певної функціональності програмної системи, що розробляється. Структура даного артефакту полягає в «трійці»: що треба зробити, очікуваний результат, фактичний результат. Безпосередньо сам тестовий випадок складається з 3 частин (рис. 2.2):

- Передумови (PreConditions) - список кроків, які призводять систему, що досліджується, в стан, придатний для тестування, або список перевірок умов того, що система вже знаходиться в необхідному стані.
- Опис (Description) - список дій, за допомогою яких здійснюється основна перевірка функціоналу (після якої і звіряється фактичний результат з очікуваним).
- Післяумови (PostConditions) - список дій, які повертають систему в початковий стан.

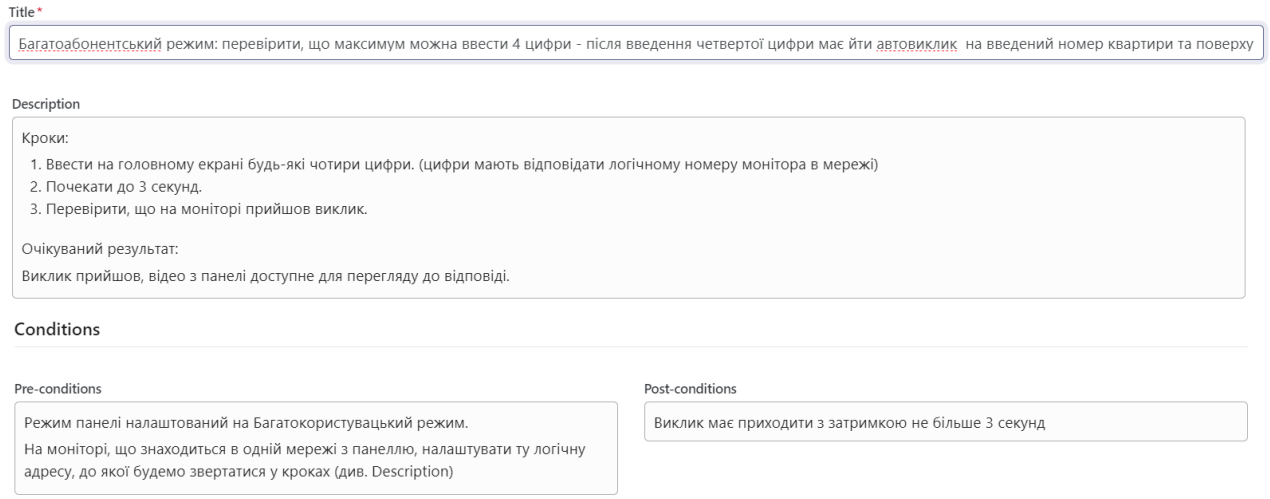


Рис. 2.2. Вигляд тест-кейсу на прикладі багатокористувацької панелі P-01 в додатку Qase

Спосіб опису тест кейсів і їх структура може в кожній компанії або команді бути різним, мати різні глибини опису необхідних дій і результатів, мати різні структурні складові. Але правильна структурованість і висока зручність шаблонів тестових

випадків, може набагато скоротити час рутинних заповнень форм і підвищити ефективність в цілому.

Тестовий набір (Тест-сьют) - це комбінація тестових сценаріїв, для перевірки певної частини програмного забезпечення, об'єднаної загальною функціональністю або цілями, що передують запуском даного набору(рис. 2.3).

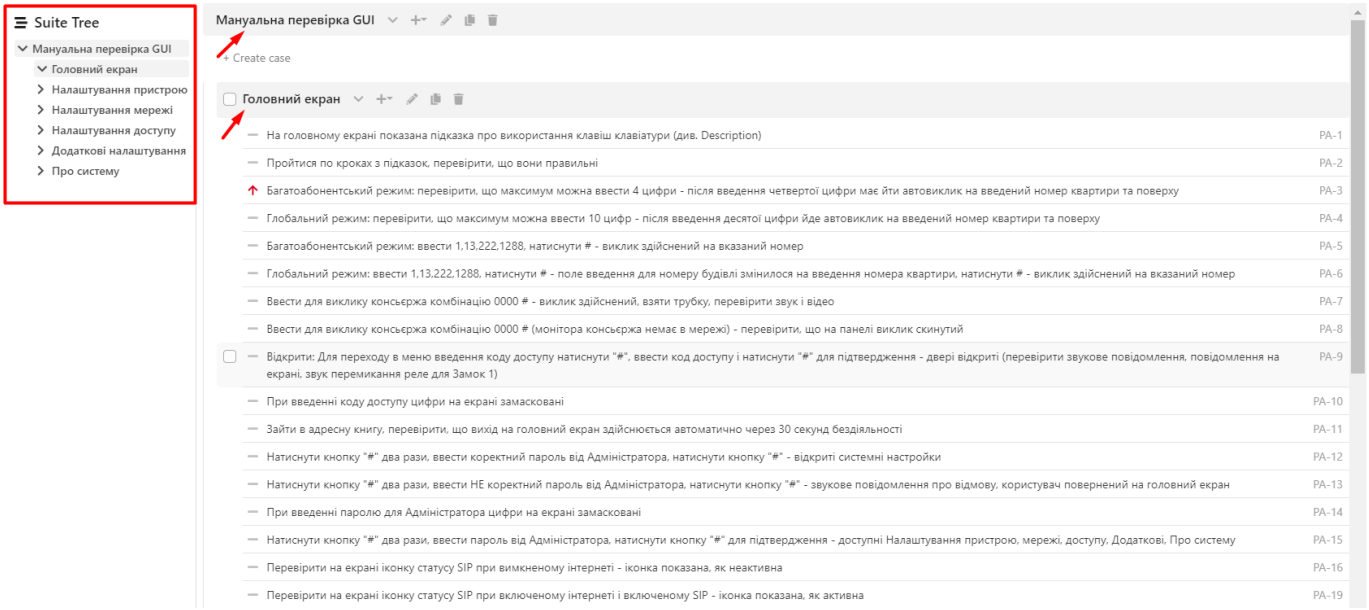


Рис. 2.3. Вигляд тест-сьютів на прикладі багатокористувацької панелі P-01 в додатку Qase

Тест-звіт (Тест-репорт) - є сумарною інформацію про проходження тестів, на основі аналізу і порівняння з очікуваними результатами виконується детальна оцінка якості продукту, що тестується і поточного статусу процесу тестування. Рекомендується записувати і зберігати результати тестування для кожного етапу, так як це є одним із найважливіших артефактів тестування. Структура такого документу дає легко оцінити статистику вразливих місці ПЗ(рис. 2.4).

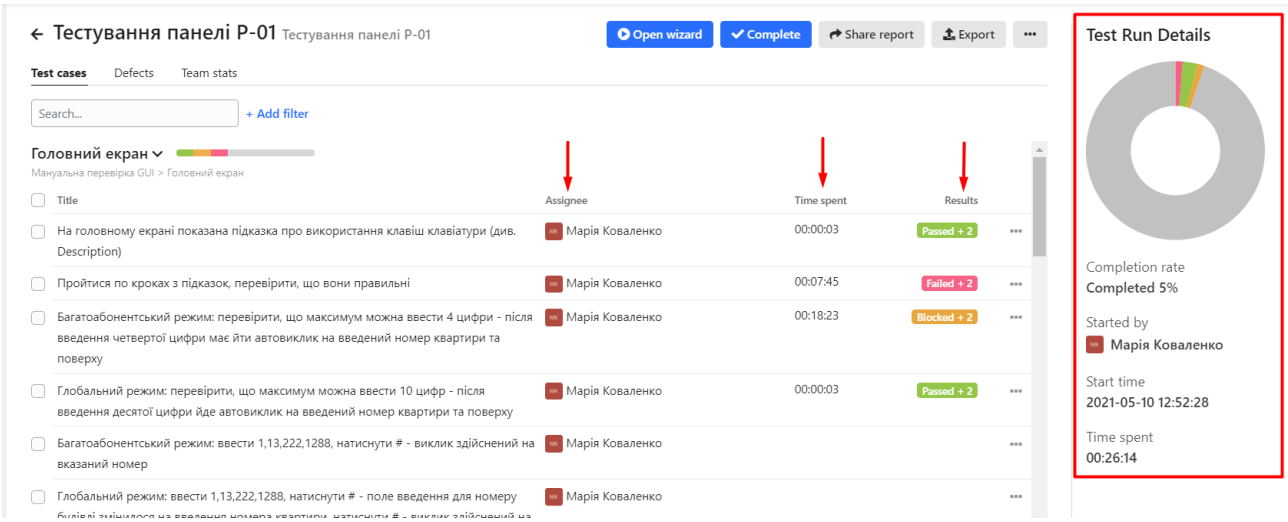


Рис. 2.4. Вигляд тест-звіту на прикладі багатокористувацької панелі P-01 в додатку Qase

2.3. Звіти про виявлені дефекти

Система управління тестуванням використовується для зберігання інформації та належного виконання тестів, визначення пріоритетів відповідно до їх графіка та отримання інформації у формі звітів про стан випробувань та якість програмного забезпечення. Інструменти кожної системи використовують різні підходи до тестування, тому містять різні набори функцій. Зазвичай такі методи роботи використовуються для планування ручного тестування, збору даних про результати існуючих контрольних списків та тестових випадків або отримання оперативної інформації у формі звітів. Системи управління тестами допомагають керувати процесом тестування та забезпечують швидкий доступ до аналізу даних, інструментів співпраці та кращої співпраці між кількома командами проектів. Залежно від інструменту керування звітів про помилки, зовнішній вигляд може дещо відрізнятися, окремі частини можна додавати або видаляти, але концепція залишається незмінною[5].

Вирішено зупинитися на виборі системи Qase - нещодавно випущений продукт, який з легкістю можна використовувати на роботі. Ця хмарна система менеджменту допомагає збільшити продуктивність та організувати зручне тестування програмного

забезпечення. Серед найкращих можливостей даної системи можна виділити наступні пункти:

- Резерв тестів: можливо організувати тести у логічні групи.
- Розробка заходів, встановлені на пріоритет та актуальність.
- Можливо встановити час, затрачений на випробування для кожного тестового випадку.
- Можливість імпортувати та експортувати проектну документацію.
- Прямий опис дефектів в інтегрованих вікнах чек-листу, що зменшує час тестування.
- Співпраця з баг-трекінговими системами, нарпиклад: JIRA, YouTrack.
- Поєднання результатів автоматизованих тестів із REST API.

Qase дозволяє швидко та зручно описувати дефекту. Загалом, є два шляхи:

1. Зайти на вкладку Дефекти та натиснути на кнопку «Створити новий дефект» (рис. 2.5).

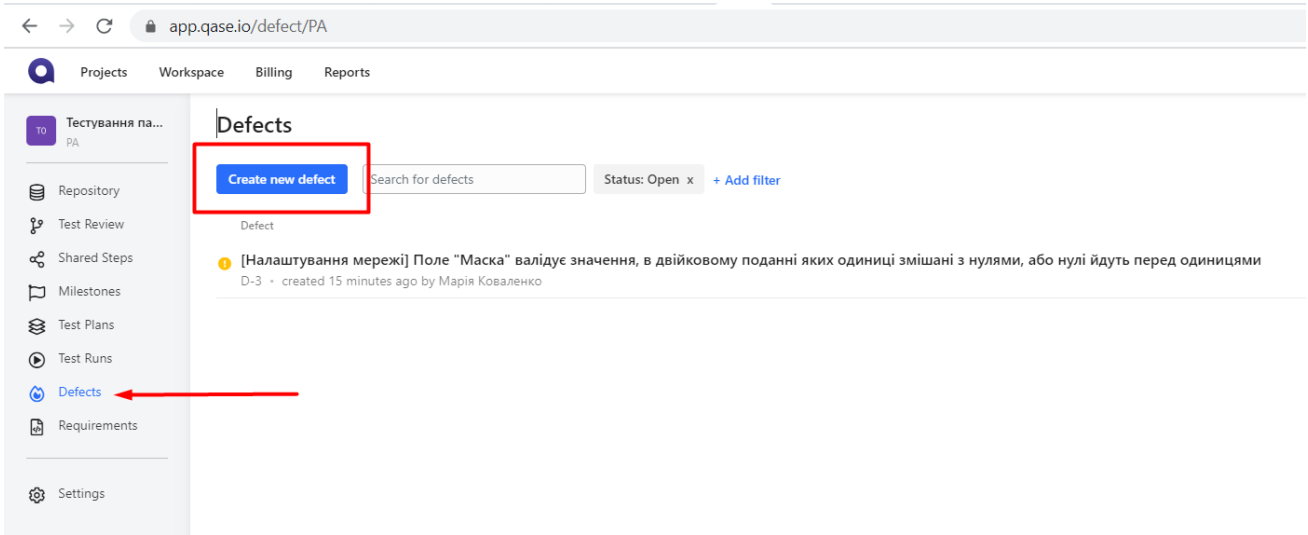


Рис. 2.5. Перший спосіб заведення багу у системі Qase

2. При проходженні чек-листу, коли тест показав негативний результат, тобто очікувана поведінка не співпадає із фактичною, натискаємо кнопку із надписом

«Failed», що свідчить про те, що результат роботи програми не відповідає вимогам (рис. 2.6).

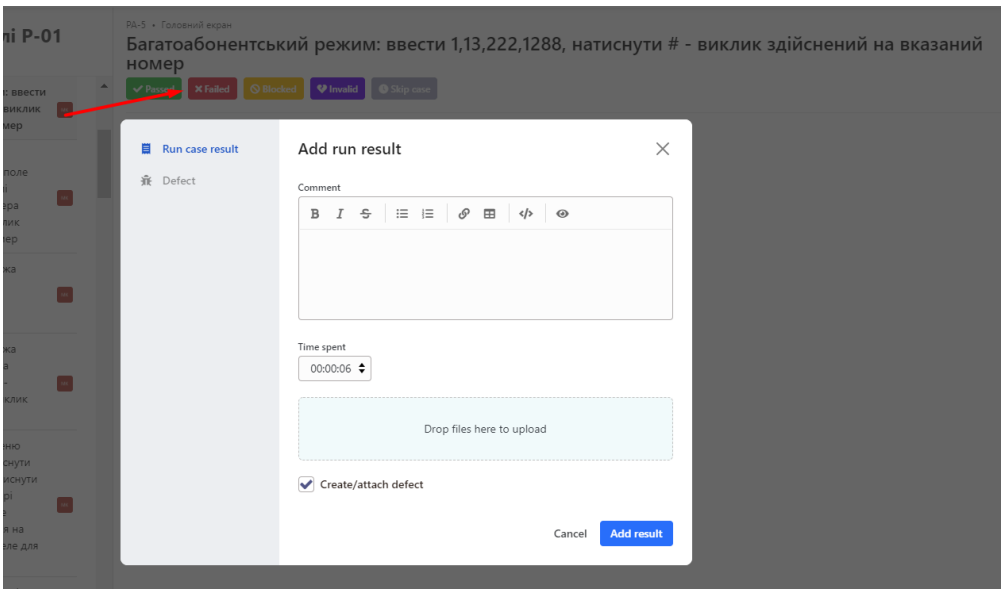


Рис. 2.6. Другий спосіб заведення багу у системі Qase

Загальний вигляд структури звіту про дефект описується атрибутами, кожен із яких має бути обов'язковим, адже через порушення архітектури баг-репорту можуть втратитись важливі дані, наприклад: версія прошивки, оточення, кроки, за якими можна відтворити даний баг або навіть його актуальну поведінку.

Ідентифікатор - це унікальне значення, яке однозначно відрізняє один звіт про помилку від іншого і використовується у всіх видах посилань. Зазвичай у цьому випадку ідентифікатором повідомлення про несправність може бути лише унікальний номер, але (якщо інструмент звітування дозволяє це) може бути набагато складніше: додаються префікси, суфікси та інші значущі компоненти, які допоможуть швидко з'ясувати сутність дефекту або до якого модуля системи стосується помилка.

Короткий опис (аналог заголовку) повинен бути вичерпним та давати відповідь на запитання три запитання: "Що сталося?", "Де сталося?"(на якій сторінці, в якому пункті меню тощо), "Коли сталося?" (тобто за яких умов). Наприклад, при перевірці багатокористувацької панелі Р-01 відсутні підказки на головному екрані, тоді

короткий опис може бути представлений як: "Відсутні підказки на головному екрані після переходу в режим налаштувань":

- Що сталося? - Відсутні підказки.
- Де сталося? - На головному екрані.
- Коли сталося? - Після переходу в режим налаштувань.

Детальний опис представляє в розгорнутому вигляді необхідну інформацію про дефект, а також обов'язково опис фактичного і очікуваного результатів і посилання на вимогу (якщо це можливо).

Кроки по відтворенню описують дії, які необхідно виконати для відтворення дефекту. Це поле схоже на кроки тест-кейсу, за винятком одного важливого відмінності: тут дії прописуються максимально докладно, із зазначенням конкретних введених значень і найдрібніших деталей, тому що відсутність цієї інформації в деяких випадках може привести до неможливості відтворення дефекту.

Важливість показує ступінь шкоди, яка завдається проекту існуванням дефекту. Пріоритетність показує, як швидко дефект повинен бути усунений.

Коментар або додаткова інформація - може містити будь-які корисні для розуміння і виправлення дефекту дані. Іншими словами, сюди можна писати все те, що не можна писати в інші поля.

Додатки – являють собою не стільки поле, скільки список, що прикріплений до звіту про дефект додатків (наприклад, знімки екран, після аварійного стану програми).

Результат виконання тесту зазвичай описується одним словом, так як деталізація некоректної роботи описується у атрибутах баг-репорту. Результат може не являти собою окреме поле, а виступати лише у ролі позначки. Наприклад, достатньо виділити кейс із чек-листу зеленим кольором для того, щоб позначити, що тест пройдений успішно або червоний, щоб показати, що виявлено дефект. В такому випадку загально можна описати такі види результатів тестових випадків:

Пройдено (Passed) – успішне виконання тестового кейсу, фактичний та очікуваний результати сходяться між собою.

Невдало пройдено (Failed) – неуспішне виконання, тестування виявило дефект, який обов’язково має бути описаним.

Заблоковано (Blocked) – тест не може бути пройденим через те що дана функція ще не реалізована або присутній баг, через який виконати кроки неможливо. (наприклад, через помилку неможливо перейти до налаштувань мережі, тоді всі тестові набори пов’язані із цим будуть у стані Blocked.

Пропущено (Skipped) – тест не може бути перевіреним на розсуд тестувальника або через брак часу, але при цьому функція реалізована та готова до перевірки[6].

Після занесення дефекту до системи, опис набув вигляду згідно рисунка 2.7, в якому показано усі використані атрибути:

Рис. 2.7. Зовнішній вигляд баг-репорту у системі Qase на прикладі багатокористувацької панелі P-01

2.4. Висновки до розділу

У ході написання даного розділу було досліджено поняття тестової документації та вимог до програмного забезпечення. Для ручного тестування монітору М-01 та багатокористувацької панелі Р-01 було проаналізовано коректність написання таких документів: чек-листи, тест-кейси, тест-сьюти та тест-репорти. Також було описано усі атрибути баг-репорту, які будуть використовуватись після закінчення тестування для опису існуючих дефектів. На прикладі системи керування тестуванням Qase було практично обґрунтовано вибір TMS, показано зовнішній вигляд тестової документації уже в системі для розуміння принципів роботи і загального огляду функціоналу.

РОЗДІЛ 3
РУЧНЕ ТЕСТУВАННЯ ПЗ

3.1. Тестування багатокористувацької панелі на прикладі моделі P-01 та монітору на прикладі моделі M-01

На початку тестування варто розробити чек-лист на перевірку, який буде уміщати в себе деяку кількість тест-кейсів та буде охоплювати певний функціонал. Для перевірки монітору, проаналізувавши кожну із вибраних технік Розділу 1, розроблено чек-лист для перевірки, що описаний у Додатку А.

Ці дані внесено у систему контролю тестування Qase, для цього створено проекти для монітору та панелі (рис. 3.1):

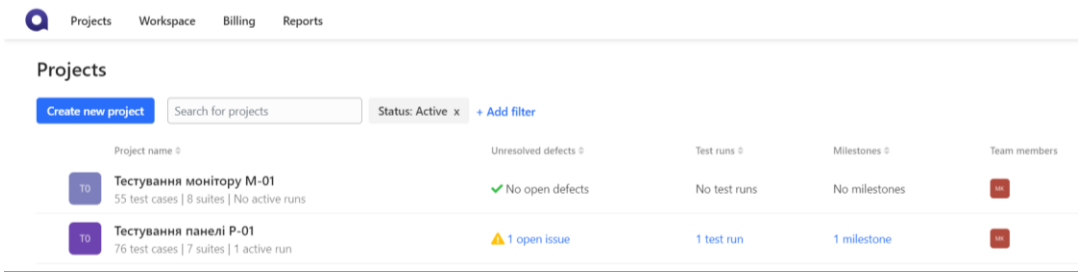


Рис. 3.1. Створення проектів в Qase TMS

Заповнено описаними тест-кейсами чек-лист проекту для панелі (рис. 3.1):

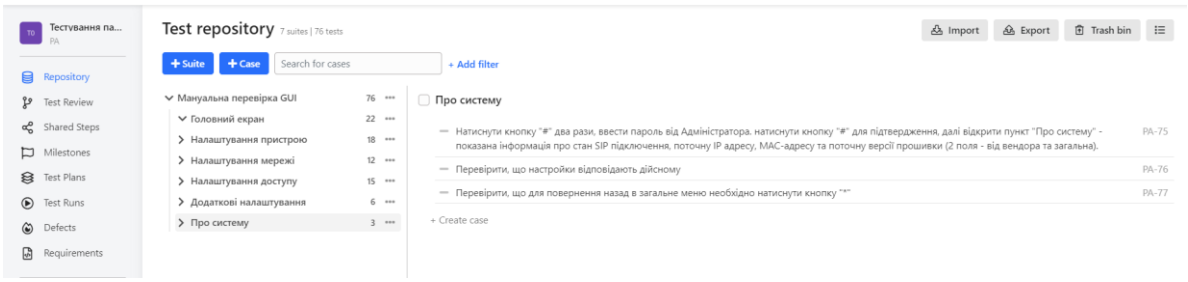


Рис. 3.2. Створений чек-лист для панелі в Qase TMS

Для перевірки деякого функціоналу монітору, вибравши відповідні техніки, що було досліджено в Розділі 1, розроблено чек-лист для перевірки, що описаний у Додатку Б.

Заповнено описаними тест-кейсами чек-лист проекту для монітору (рис. 3.3):

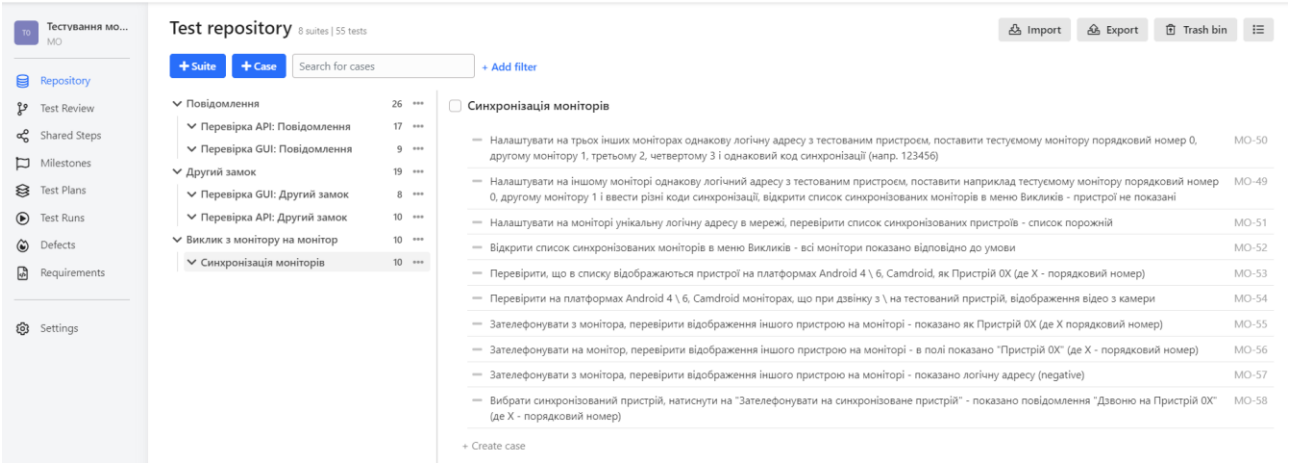


Рис. 3.3. Створений чеклист для панелі в Qase TMS

Створено тест-плани для проектів (рис. 3.4, 3.5):

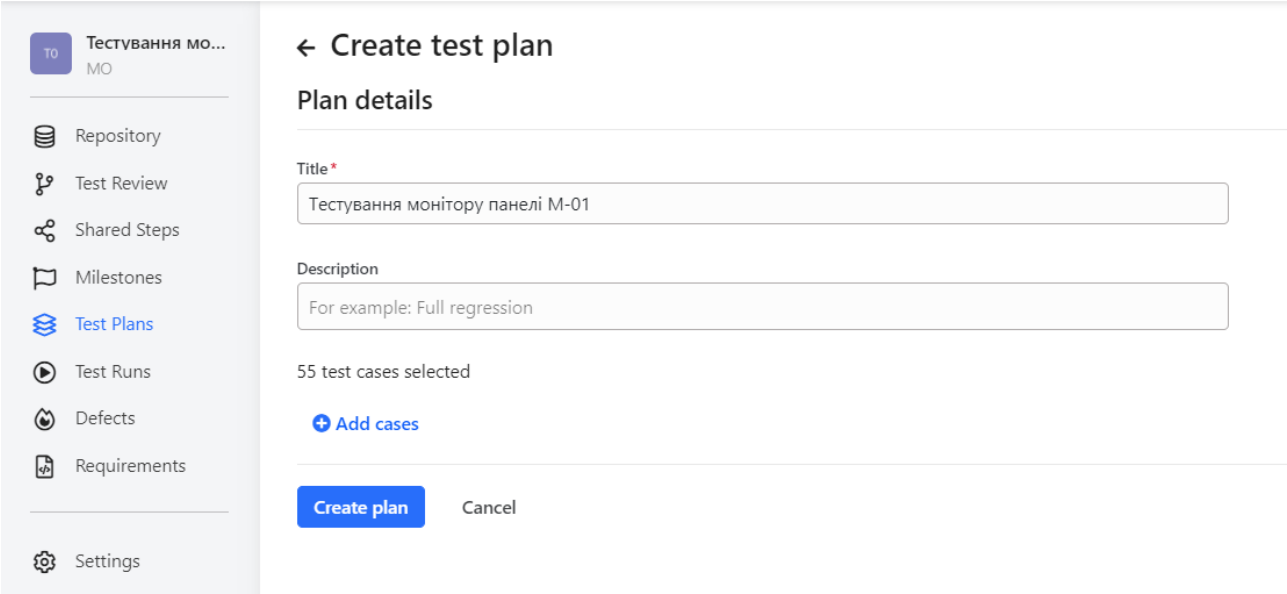


Рис. 3.4. Створення тест-плану для монітору в Qase TMS

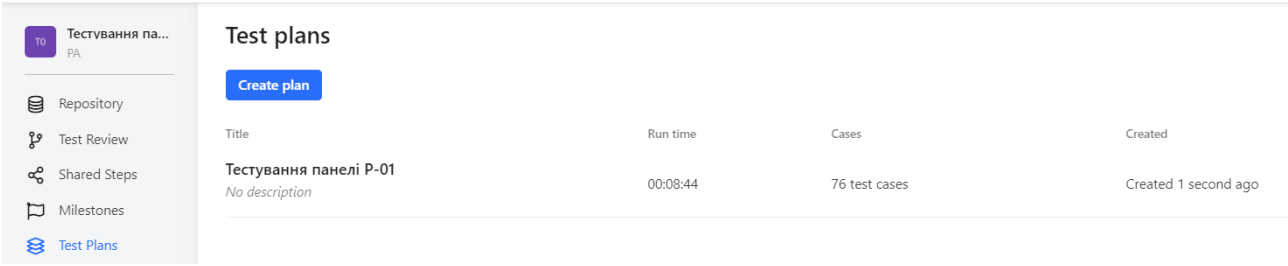


Рис. 3.5. Створений тест-план для панелі в Qase TMS

У ході тестування позначено статус кожного тест-кейсу за допомогою спеціальних кнопок(рис. 3.6):

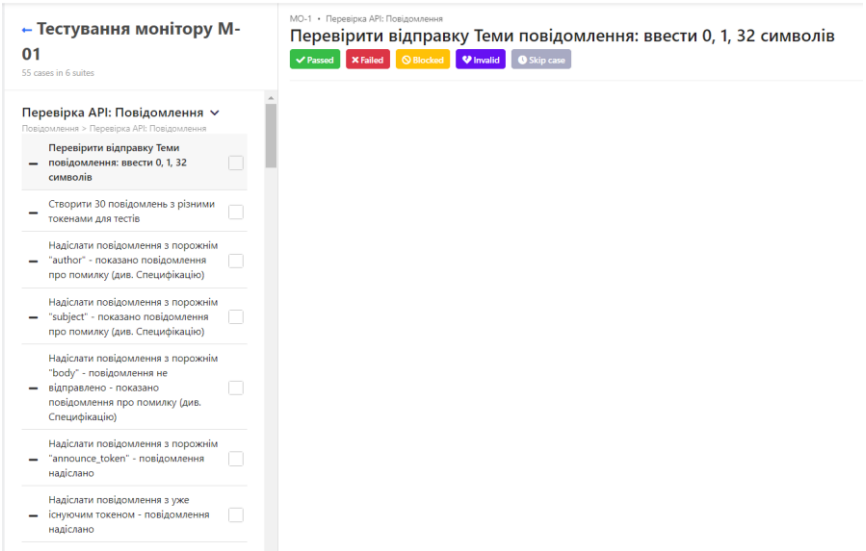


Рис. 3.6. Створений тест-план для панелі в Qase TMS

Після завершення тестування зроблено тест-звіт по чек-листу, кожному тестовому набору та випадку, витрачений на тестування час, дату початку та статистику (рис. 3.7):

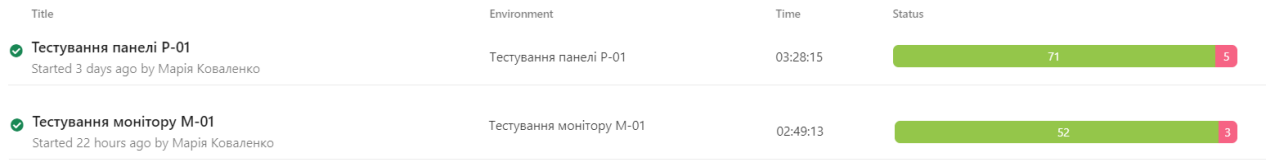


Рис. 3.7. Створений тест-звіт для панелі та монітору в Qase TMS

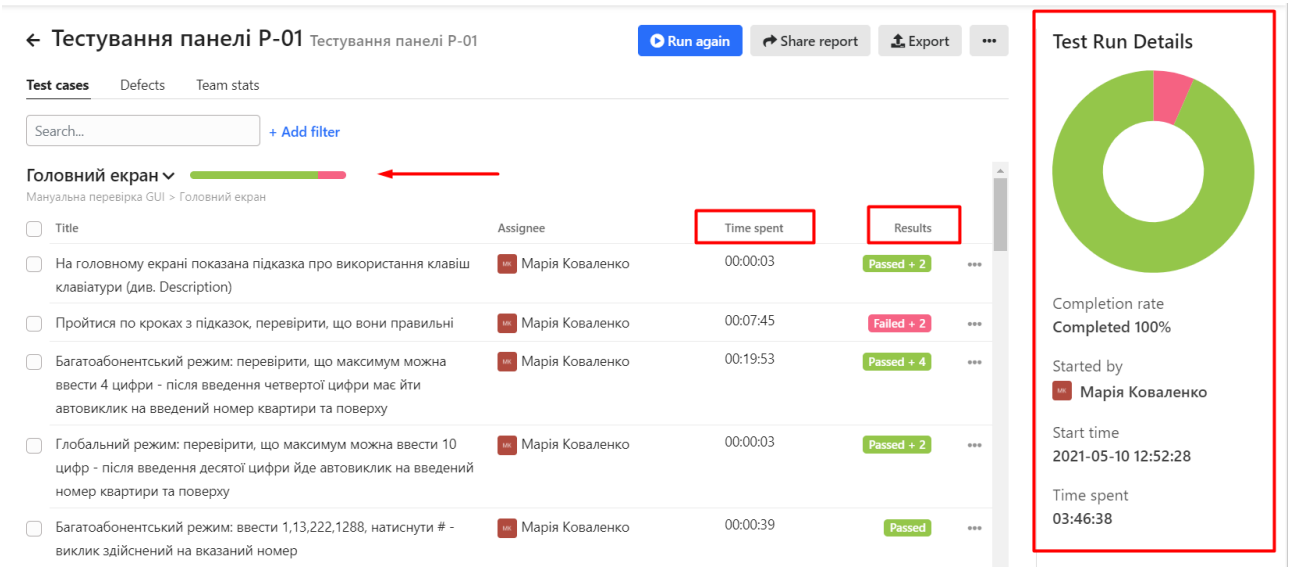


Рис. 3.8. Детальний тест-звіт на прикладі панелі в Qase TMS

3.2. Заведення звітів про виявлені дефекти

Деякі результати перевірки були негативними і згідно вже відомих даних із Розділів 1 та 2 описано кожний баг репорт та внесено їх в Qase TMS:

1. На головному екрані відсутнє повідомлення «Конфлікт MAC адрес», якщо змінити IP на статичне значення аналогічно іншому пристрою в мережі.

Оточення:

GUI панелі P-01

Версія прошивки 1.7.0

Передумови:

Налаштувати на іншому пристрої в мережі IP-адресу наприклад 192.168.88.222

Кроки для відтворення:

1. Ввести пароль від Адміністратору.
2. Зайти в налаштування мережі.
3. Налаштувати на панелі адресу, аналогічну в передумові, напр. 192.168.88.222.
4. Зайти на Головний екран.
5. Почекати до 1хв.

Очікуваний результат:

Повідомлення «Конфлікт MAC адрес» має з’являтись одразу, як тільки користувач повертається на Головний екран. Знаходження не екрані – правий кут зверху.

Фактичний результат:

Повідомлення відсутнє на екрані.

Вкладення (рис 3.9):

Рис. 3.9. Вкладення до баг-репорту №1

2. Іконка статусу реєстрації SIP показана, як активна при не підключеному Ethernet-кабелю.

Оточення:

GUI панелі P-01

Версія прошивки 1.7.0

Кроки для відтворення:

1. Зайти на головний екран.
2. Висунути з панелі Ethernet-кабель.

Очікуваний результат:

Іконка SIP перекреслена (неактивна).

Фактичний результат:

Іконка SIP не перекреслена (активна).

Вкладення (рис 3.10):

Рис. 3.10. Вкладення до баг-репорту №2

3. Іконка статусу реєстрації SIP показана, як активна при вимкненому SIP.

Оточення:

GUI панелі P-01

Версія прошивки 1.7.0

Кроки для відтворення:

1. Ввести пароль від Адміністратору.

2. Зайти в налаштування мережі.
3. Вимкнути SIP.
4. Зайти на головний екран.

Очікуваний результат:

Іконка SIP перекреслена (неактивна).

Фактичний результат:

Іконка SIP не перекреслена (активна).

Вкладення (рис 3.11):

Рис. 3.11. Вкладення до баг-репорту №2

4. Повідомлення «Виклик завершено» не зникає після дзвінку на монітор консьєржа, який відсутній в мережі.

Оточення:

GUI панелі P-01

Версія прошивки 1.7.0

Передумови:

Монітору консьєржа немає в мережі.

Кроки для відтворення:

1. Ввести для виклику консьєржа комбінацію 0000# або натиснути кнопку «Консьєрж».

Очікуваний результат:

На панелі виклик скинутий, показане повідомлення «Виклик завершено», яке автоматично через 3 секунди зникає і користувач повертається на Головний екран.

Фактичний результат:

На панелі виклик не скинутий, показане повідомлення «Виклик завершено», яке автоматично не повертає користувача на Головний екран.

5. [Налаштування мережі] Поле "Маска" валідує значення, в двійковому поданні яких одиниці змішані з нулями, або нулі йдуть перед одиницями

Оточення:

GUI панелі P-01

Версія прошивки 1.7.0

Кроки для відтворення:

1. Відкрити Налаштування на панелі P-01
2. Зайти в розділ меню Налаштування мережі.
3. Ввести коректний пароль адміністратора, натиснути "Вхід".
4. В поле "Маска" вписати значення, в двійковому поданні якого одиниці змішані з нулями, або нулі йдуть перед одиницями (див. Примітку)
5. Натиснути Зберегти.

Примітка:

Приклади:

255.0.255.0 (11111111.00000000.11111111.00000000)

120.22.123.12 (01111000.00010110.01111011.00001100)

Очікуваний результат:

Поле "Маска" валідує тільки ті значення, двійкове подання яких є послідовністю N одиниць зліва, і 32-N нулів після. Варто врахувати, що запис повинен бути в форматі десяткових чисел з точками, як і реалізовано зараз. При введенні невірної маски і натисканні "Зберегти" - звукове повідомлення про помилку, "неправильна" маска не збережена.

Фактичний результат:

При введенні невірної маски і натисканні "Зберегти" - звукове повідомлення про успішність змінених налаштувань. Якщо перезайти в меню налаштування мережі - бачимо, що введена "неправильна" маска не збережена і відповідає минулому встановленому значенню.

Звіти про дефекти після тестування монітору:

1. Логічну адресу, замість номеру монітору, показано на екрані іншого пристрою при дзвінку з М-01.

Оточення:

GUI монітору М-01

Версія прошивки 2.4.1

Передумови:

1. Налаштувати однакову логічну адресу на двох моніторах.
2. Поставити порядковий номер першому монітору 1, а другому 2.
3. Ввести однаковий код синхронізації.

Кроки для відтворення:

1. Зайти в Меню викликів.
2. Зайти в список синхронізованих пристроїв.
3. Зателефонувати з монітора, натиснувши кнопку Виклик.
4. Перевірити відображення іншого пристрою на моніторі.

Очікуваний результат:

При дзвінку на іншому пристрої показано «Пристрій 0X (де X – вписаний порядковий номер).

Фактичний результат:

При дзвінку на іншому пристрої показано логічну адресу монітору.

2. Параметри повідомлення у Меню повідомлень оновлюються після перезавантаження монітору.

Оточення:

GUI монітору М-01

Версія прошивки 2.4.1

Передумови:

1. Надіслати повідомлення з токеном, наприклад «04e0e20c-6d7f-4468-8be0-bcbb19796a», параметри author, body, type, subject довільні та відповідають специфікації.

Кроки для відтворення:

1. Описати повідомлення з токеном, аналогічним передумові («04e0e20c-6d7f-4468-8be0-bcbb19796a»)
2. В цьому ж повідомлення ввести параметри author, body, type, subject не відповідають даним в передумові та відповідають специфікації.
3. Надіслати повідомлення.
4. Зайти в Меню повідомлень на моніторі.

Очікуваний результат:

Отримано статус 200 ОК, дані з передумови оновились на дані з кроку 2.

Фактичний результат:

Отримано статус 200 ОК, але дані з передумови оновлюються на дані з кроку 2 тільки після перезавантаження монітору.

3. Кнопка назад не повертає користувача на головний екран із Меню повідомлень.

Оточення:

GUI монітору M-01

Версія прошивки 2.4.1

Кроки для відтворення:

1. Зайти на головний екран.
2. Зайти до меню викликів.
3. Натиснути кнопку назад (Будиночок у правому кутку зверху)

Очікуваний результат:

Користувач потрапляє на головний екран.

Фактичний результат:

Кнопка неактивна. При натисненні – нічого не відбувається.

Кожен дефект опишемо у TMS Qase, призначимо його важливість до виконання (рис. 3.12):

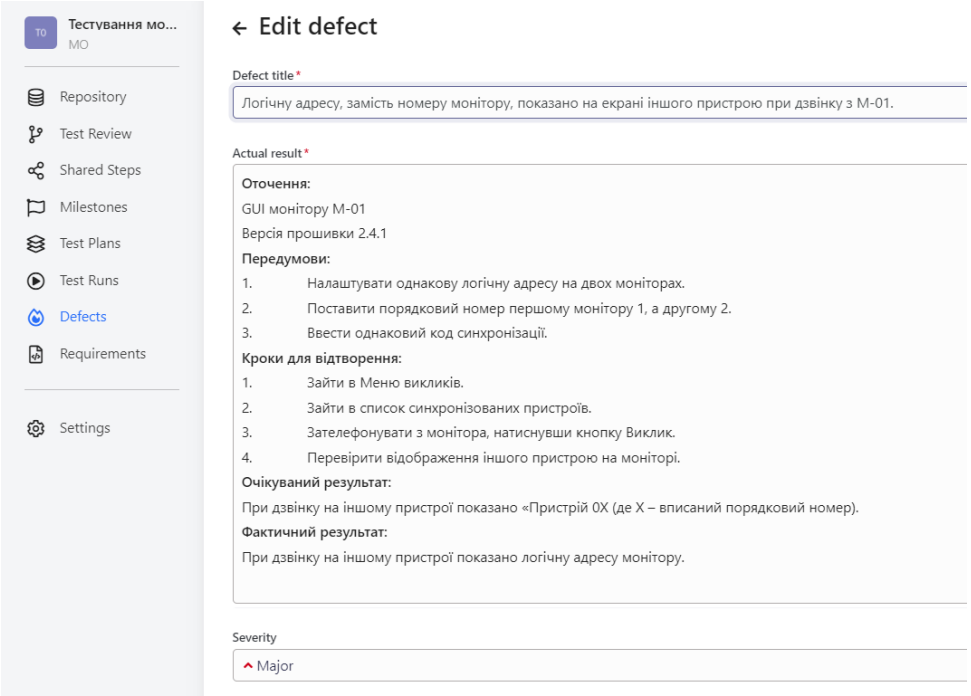


Рис. 3.12. Приклад опису баг-репорту №1 для монітору

Після описання дефектів, створено загальний список відкритих багів по кожному проекту (рис. 3.13, 3.14):

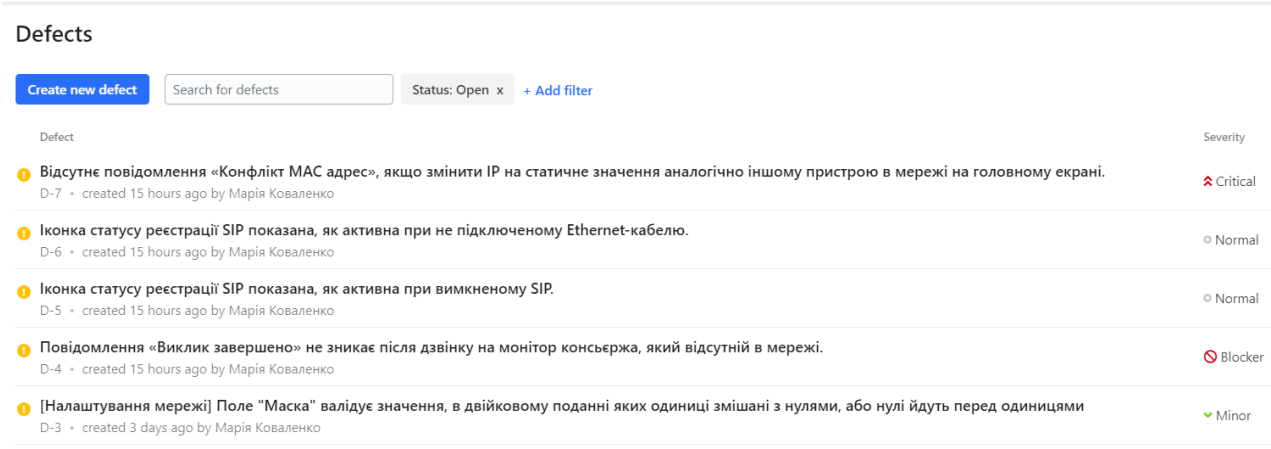


Рис. 3.13. Список помилок після тестування панелі P-01

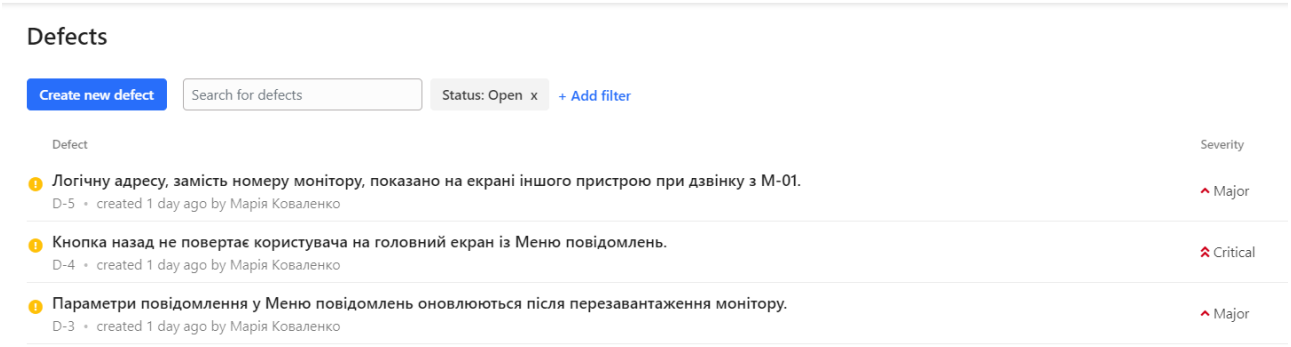


Рис. 3.14. Список помилок після тестування монітору M-01

3.3. Висновок до розділу

В ході виконання роботи над створенням розділу III, написано чек-лист на перевірку функціоналу монітору M-01 та панелі P-01, зроблене тестування пристроїв, проаналізовано статистику результатів. Також створено баг-репорти на основі виявлених неспівпадінь очікуваного та фактичного результату після тестування ПЗ. Всі дані були занесені у хмарне сховище Qase для зручного управління процесу перевірки продукту.

ВИСНОВКИ

В результаті виконання дипломної роботи протестоване програмне забезпечення фірми “YouIP”, а саме: монітор M-01 та багатокористувацька панель P-01.

У першому розділі сформульовано характеристику задачі, мету її вирішення та цілі, які вона переслідує, проведено аналіз існуючих видів та підходів до тестування ПЗ. Досліджено техніки тест-дизайну програмного продукту. Проаналізовано поняття багу та його життєвий цикл.

У другому розділі досліджено поняття тестової документації та вимог до програмного забезпечення. Також розглянуто атрибути баг-репорту для опису існуючих дефектів, які будуть описуватись в останньому розділі після закінчення тестування.

У третьому розділі реалізовано список на перевірку функціоналу, зроблене тестування пристроїв і по негативним результатам створено баг-репорти. Зроблено тестування та звітності за допомогою додатку Qase.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Савин Р. Тестирование DOT COM./ Р. Савин - Москва: Дело, 2007. – 312 с.
2. Куликов С. Тестирование программного обеспечения. Базовый курс./ С. Куликов - Минск: Четыре четверти, 2017. – 310 с.
3. Бейзер Б. Тестирование черного ящика./ Б. Бейзер – Санкт-Петербург: Питер, 2004. – 314 с.
4. Карл И. Разработка требований к программному обеспечению./ И. Карл - Москва: Русская редакция, 2004. – 576 с.
5. Рекс Б. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование./ Б. Рекс - Москва: Лори, 2017. – 544 с.
6. Старолетов С. Основы тестирования и верификации программного обеспечения./ С. Старолетов - Санкт-Петербург: Лань, 2020. – 344 с.

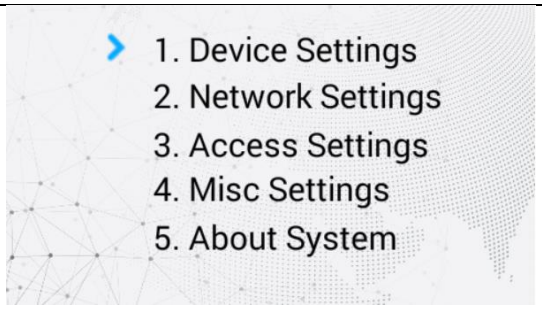
ДОДАТКИ

Додаток А

Чек-лист до тестування панелі P-01

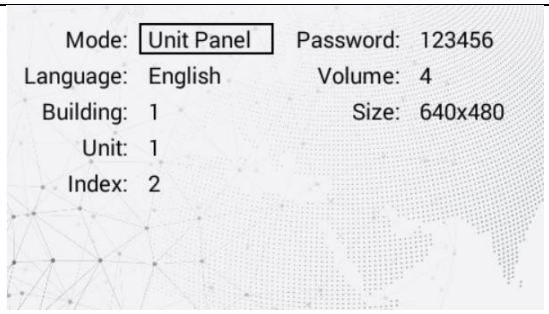
ІД	Короткий опис	Опис	Тест-сьют
1	На головному екрані показана підказка про використання клавіш клавіатури (див. Description)		Головний екран
2	Пройтися по кроках з підказок, перевірити, що вони правильні		Головний екран
3	Багатоабонентський режим: перевірити, що максимум можна ввести 4 цифри - після введення четвертої цифри має йти автовиклик на введений номер квартири та поверху		Головний екран
4	Глобальний режим: перевірити, що максимум можна ввести 10 цифр - після введення десятої цифри йде автовиклик на введений номер квартири та поверху		Головний екран
5	Багатоабонентський режим: ввести 1,13,222,1288, натиснути # - виклик здійснений на вказаний номер		Головний екран

6	Глобальний режим: ввести 1,13,222,1288, натиснути # - поле введення для номеру будівлі змінилося на введення номера квартири, натиснути # - виклик здійснений на вказаний номер		Головний екран
7	Ввести для виклику консьєржа комбінацію 0000 # - виклик здійснений, взяти трубку, перевірити звук і відео		Головний екран
8	Ввести для виклику консьєржа комбінацію 0000 # (монітора консьєржа немає в мережі) - перевірити, що на панелі виклик скинутий		Головний екран
9	Відкрити: Для переходу в меню введення коду доступу натиснути "#", ввести код доступу і натиснути "#" для підтвердження - двері відкриті (перевірити звукове повідомлення, повідомлення на екрані, звук перемикання реле для Замок 1)		Головний екран
10	При введенні коду доступу цифри на екрані замасковані		Головний екран

11	Зайти в адресну книгу, перевірити, що вихід на головний екран здійснюється автоматично через 30 секунд бездіяльності		Головний екран
12	Натиснути кнопку "#" два рази, ввести коректний пароль від Адміністратора, натиснути кнопку "#" - відкриті системні настройки		Головний екран
13	Натиснути кнопку "#" два рази, ввести НЕ коректний пароль від Адміністратора, натиснути кнопку "#" - звукове повідомлення про відмову, користувач повернений на головний екран		Головний екран
14	При введенні паролю для Адміністратора цифри на екрані замасковані		Головний екран
15	Натиснути кнопку "#" два рази, ввести пароль від Адміністратора, натиснути кнопку "#" для підтвердження - доступні Налаштування пристрою, мережі, доступу, Додаткові, Про систему		Головний екран

16	Перевірити на екрані іконку статусу SIP при вимкненому інтернеті - іконка показана, як неактивна		Головний екран
17	Перевірити на екрані іконку статусу SIP при вимкненому SIP - іконка показана, як неактивна		Головний екран
18	Перевірити на екрані іконку статусу SIP при неправильному логіну та паролю від SIP-аккаунта - іконка показана, як неактивна		Головний екран
19	Перевірити на екрані іконку статусу SIP при включеному інтернеті і включеному SIP - іконка показана, як активна		Головний екран
20	Перевірити на екрані іконку статусу в локальній мережі при підключеному до Ethernet, але вимкненому в ньому Internet-лінку - іконка показана, як активна		Головний екран

Продовження Додатку А

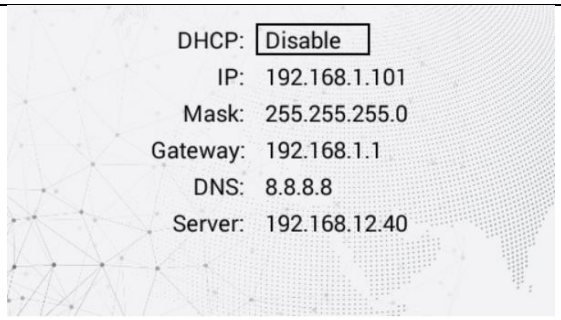
21	Перевірити на екрані іконку статусу в локальній мережі при підключеному до Ethernet і включеному інтернеті - іконка показана, як активна		Головний екран
22	Перевірити на екрані іконку статусу в локальній мережі при не підключеному Ethernet-кабелю - іконка показана, як неактивна		Головний екран
23	Натиснути кнопку "#" два рази, ввести пароль від Адміністратора. натиснути кнопку "#" для підтвердження, далі відкрити налаштування пристрою - доступні налаштування Режим / Мова / Будівля / Парадне / Пароль / Гучність		Налаштування пристрою
24	Режим: Натиснути на поле два рази за допомогою клавіші # - звук успішно виконаної команди, перевірити на панелі, що режим залишився таким, як і був		Налаштування пристрою
25	В поле режим показаний поточний режим пристрою		Налаштування пристрою

Продовження Додатку А

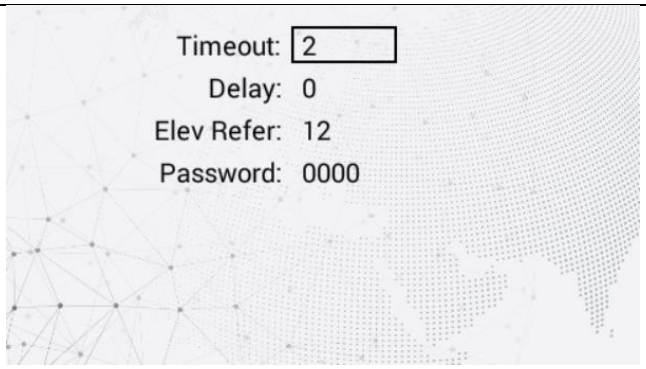
26	Вибір мови пристрою English, зберегти - перевірити, що мова змінився на панелі		Налаштування пристрою
27	Вибір мови пристрою Ukrainian, зберегти - перевірити, що мова змінився на панелі		Налаштування пристрою
28	В поле мова показана поточна мова пристрою		Налаштування пристрою
29	Натиснути # на поле Номер будинку, ввести 1, 100, 9999, зберегти за допомогою # - перевірити, що значення застосувалися, звук успішно виконаної команди		Налаштування пристрою
30	Натиснути # на поле Номер парадного, ввести 0, 20,99, зберегти за допомогою # - звук успішно виконаної команди		Налаштування пристрою
31	Натиснути # на поле Порядковий номер, ввести 0, 20,99, зберегти за допомогою # - перевірити, що значення застосувалися, звук успішно виконаної команди		Налаштування пристрою
32	Вибрати режим Глобальний, поля Парадне і Будівля приймають значення 00		Налаштування пристрою

Продовження Додатку А

33	Вибрати режим Багатоабонентський, поля Парадне і Будівля приймають стандартні або раніше збережені користувачем значення		Налаштування пристрою
34	Пароль: Змінити пароль на 1234 - перевірити вхід по новому паролю в налаштування панелі - налаштування відкриті успішно		Налаштування пристрою
35	Пароль: Змінити пароль на 1234 - перевірити вхід по старому паролю в налаштування панелі - налаштування не відкриті, користувач повертається на головний екран, звукове повідомлення про відмову		Налаштування пристрою
36	Поточний пароль показаний в полі		Налаштування пристрою
37	Вибрати рівень гучності: 1,3,6, зберегти - перевірити, що рівень змінився на панелі		Налаштування пристрою
38	Вибрати рівень гучності: 640 × 480, 1280 × 720, зберегти - перевірити, що відео доступне і змінилось на панелі (наприклад, через VLC, що якість відео відповідає установленому)		Налаштування пристрою

39	Перевірити, що для підтвердження значень необхідно натиснути кнопку "#"		Налаштування пристрою
40	Перевірити, що для повернення назад необхідно натиснути кнопку "*"		Налаштування пристрою
41	Натиснути кнопку "#" два рази, ввести пароль від Адміністратора, натиснути кнопку "#" для підтвердження, далі відкрити налаштування мережі - доступні налаштування DHCP / IP / Маска / Шлюз / DNS / Сервер		Налаштування мережі
42	Змінити IP на статичне значення аналогічно інший панелі в мережі - на головному екрані повідомлення: Конфлікт MAC адресів		Налаштування мережі
43	Перевірити валідацію поля IP: 223.255.255.254; 127.1.2.1; 192.168.1.12		Налаштування мережі
44	Перевірити валідацію поля IP: 192.168.1 .; 192.43.88 .; 0.0.0.0; порожнє поле (negative)		Налаштування мережі

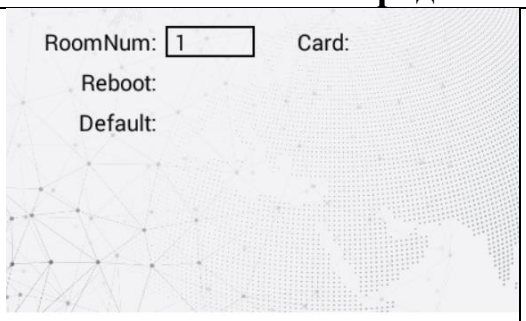
45	Перевірити валідацію поля Маска: 223.255.255.254; 255.0.0.0; 255.128.0.0		Налаштування мережі
46	Перевірити валідацію поля Маска: 223.255.0.254; 120.22.123.12; 0.0.0.0; 255.255.0. (Negative)		Налаштування мережі
47	Перевірити валідацію поля Шлюз: 147.100.100.100; 192.188.1.12		Налаштування мережі
48	Перевірити валідацію поля Шлюз: 147.100.100 .; 0.0.0.0 (negative)		Налаштування мережі
49	Перевірити валідацію поля DNS: 8.8.8.8; 192.168.1.12		Налаштування мережі
50	Перевірити валідацію поля DNS: 0.0.0.0; 192.168.1 .; 192.43.88. (Negative)		Налаштування мережі

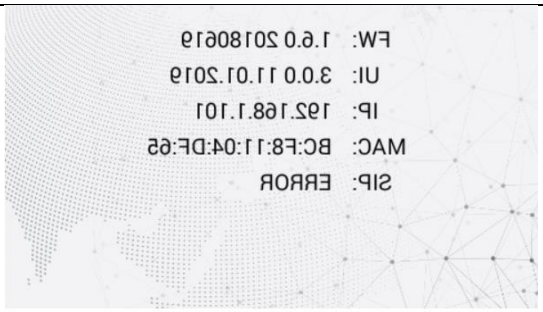
51	Перевірити, що для підтвердження значень необхідно натиснути кнопку "#"		Налаштування мережі
52	Перевірити, що для повернення назад в загальне меню необхідно натиснути кнопку "*"		Налаштування мережі
53	Натиснути кнопку "#" два рази, ввести пароль від Адміністратора. натиснути кнопку "#" для підтвердження, далі відкрити налаштування доступу - доступні налаштування Час відкриття / Затримка відкриття / Ліфт / Пароль		Налаштування доступу
54	Перевірити валідацію для часу відкриття (1-300): ввести 1, 60, 120,240,300		Налаштування доступу

Продовження Додатку А

55	Перевірити валідацію для часу відкриття (1-300): ввести 1, 60, 120,240,300		Налаштування доступу
56	Перевірити валідацію для часу відкриття (1-300): негативні значення, 0, порожнє введення, 301, символи, спецсимволи (negative)		Налаштування доступу
57	Час відкриття замку # 1-2: 1 сек - перевірити, що час, на який двері будуть відкриті, дорівнює 1 сек		Налаштування доступу
58	Час відкриття замку # 1-2: 300 сек - перевірити, що час, на яке двері буде відкрита, дорівнює 300 сек		Налаштування доступу
59	Перевірити валідацію для часу затримки перед відкриттям (0-300): ввести 0, 60, 120,240,300		Налаштування доступу
60	Затримка перед відкриттям замку # 1-2: 300 сек - перевірити, що час через який будуть відкриті двері, дорівнює 300 сек		Налаштування доступу
61	Перевірити з різними таймингами час відкриття і затримки: 1-9 \ 2-7 \ 8-3 \ 5-5 \ 9-1		Налаштування доступу
62	Перевірити валідацію поля "Номер поверху управління ліфтом" (0-98): 0, 22, 98, 09		Налаштування доступу

63	Перевірити валідацію поля "Номер поверху управління ліфтом" (0-98): мінусове значення, порожній введення, 99, символи, спецсимволи (negative)		Налаштування доступу
64	Перевірити валідацію для Загальний код відкриття (1-8 цифр): ввести 0, 12,234,4567, 13409897		Налаштування доступу
65	Перевірити валідацію для Загальний код відкриття (1-8 цифр): порожнє введення, 9 цифр, пробіли, літери, спецсимволи (negative)		Налаштування доступу
66	Пароль: Для переходу в режим введення паролю натиснути клавішу "#", ввести пароль, напр. 0000 і повторно натиснути "#"		Налаштування доступу
67	Перевірити, що "*" скасовує введення		Налаштування доступу
68	Перевірити, що для повернення назад в загальне меню необхідно натиснути кнопку "*"		Налаштування доступу

69	Натиснути кнопку "#" два рази, ввести пароль від Адміністратора, натиснути кнопку "#" для підтвердження, далі відкрити додаткові налаштування - доступні налаштування Квартири / Перезавантаження / Заводські / Карта		Додаткові налаштування
70	Ввести номер квартири, напр. 123 і піднести картку до області зчитувача - панель має видати відповідний звуковий сигнал, що підтверджує те, що карта зареєстрована і на дисплеї в поле "Карта" відобразиться номер піднесеної карти		Додаткові налаштування
71	Піднести додану карту до області зчитувача - двері відкриті (вітальне повідомлення на екрані і звукове вітання)		Додаткові налаштування
72	Перезавант.: Ввести цифру "1" і натиснути клавішу "#" - панель перезавантажилась		Додаткові налаштування
73	Заводські: Ввести цифру "1" і натиснути клавішу "#" - налаштування скинуті до заводських		Додаткові налаштування

74	Перевірити, що для повернення назад в загальне меню необхідно натиснути кнопку "*"		Додаткові налаштування
75	Натиснути кнопку "#" два рази, ввести пароль від Адміністратора. натиснути кнопку "#" для підтвердження, далі відкрити пункт "Про систему" - показана інформація про стан SIP підключення, поточну IP адресу, MAC-адресу та поточну версії прошивки (2 поля - від вендора та загальна).		Про систему
76	Перевірити, що налаштування відповідають дійсному		Про систему
77	Перевірити, що для повернення назад в загальне меню необхідно натиснути кнопку "*"		Про систему

Додаток Б

Чек-лист до тестування монітору М-01

ІД	Короткий опис	Опис	Тест-сьют
1	Перевірити відправку Теми повідомлення: ввести 0, 1, 32 символів		Перевірка API: Повідомлення

2	Створити 30 повідомлень з різними токенами для тестів	Приклад:
		{
		"author": "e",
		"body": "1",
		"type": "info",
		"subject": "r",
		"announce_token": "04e0e20c-6d7f-4468-8be0-bcbb19796a"
		}
			Перевірка API: Повідомлення

3	Надіслати повідомлення з порожнім "author" - показано повідомлення про помилку (див. Специфікацію)		Перевірка API: Повідомлення
4	Надіслати повідомлення з порожнім "subject" - показано повідомлення про помилку (див. Специфікацію)		Перевірка API: Повідомлення

5	Надіслати повідомлення з порожнім "body" - повідомлення не відправлено - показано повідомлення про помилку (див. Специфікацію)		Перевірка API: Повідомлення
6	Надіслати повідомлення з порожнім "announce_token" - повідомлення надіслано		Перевірка API: Повідомлення
7	Надіслати повідомлення з уже існуючим токеном - повідомлення надіслано		Перевірка API: Повідомлення
8	Надіслати повідомлення з уже існуючим токеном і перевірити, що текст повідомлення оновився		Перевірка API: Повідомлення
9	Надіслати повідомлення з ще не існуючим токеном і перевірити, що створено нове повідомлення		Перевірка API: Повідомлення
10	Перевірити відправку Тіла повідомлення: ввести 0, 1, 99, 420 символів		Перевірка API: Повідомлення

Продовження Додатку Б

11	Тіло повідомлення обмежено в 420 символів і зайві символи відсікаються: пробіл, кілька пробілів, порожнє введення, 421, 1000 символів (negative)		Перевірка API: Повідомлення
12	Перевірити, що Тема повідомлення обмежена в 32 символи і зайві символи відсікаються: ввести пробіл, порожнє введення, 33 символи (negative)		Перевірка API: Повідомлення
13	Ввести в body \ subject повідомлення скрипт <script> alert ("Hello, world!") </ Alert> - текст показаний в повідомленні на моніторі, робота коректна		Перевірка API: Повідомлення
14	Ввести в body \ subject повідомлення дати в різних форматах: 02.02.2121, 6/31 / 17- текст показаний в повідомленні на моніторі, робота коректна		Перевірка API: Повідомлення

Продовження Додатку Б

15	Ввести в body \ subject повідомлення час в різних форматах: 02.40 p.m., 03.15 p.m., 11 a.m., 23:15, 3:56, 09: 00- текст показаний в повідомленні на моніторі, робота коректна		Перевірка API: Повідомлення
16	Ввести в body \ subject повідомлення символи, які технічно не відносяться до ASCII - реалізовані символи відображені в повідомленні на моніторі коректно, які не підтримуються - повинні відсікатися		Перевірка API: Повідомлення
17	Зробити навантажувальний тест для subject and body - має пропустити тільки 32 символи для subject і 420 для body		Перевірка API: Повідомлення
18	Створити 30 повідомлень, зайти в в Повідомлення, а потім перевірити, що прийшли всі 30 повідомлень		Перевірка GUI: Повідомлення

19	Створити 30 повідомлень, перевірити, що повідомлення можна перегорнути за опомогою кнопки Вправа, щоб побачити інші		Перевірка GUI: Повідомлення
20	Натиснути на повідомлення - перевірити, що тема і тіло відображені коректно		Перевірка GUI: Повідомлення
21	Час, День, Дата показані в плейсхолдері повідомлення (список повідомлень\окреме повідомлення)		Перевірка GUI: Повідомлення
22	Надіслати повідомлення з уже існуючим токеном і перевірити, що повідомлення змінено в інтерфейсі	У моніторі тільки для першого замку, якщо відставити стандартне значення то буде відправлятися xml. Для перевірки логів необхідно підключити панель, на якій будуть тестуватися замки, по USB до робочої машині. На робочій машині встановити Android Studio, в розділі Logcat зробити перевірку.	Перевірка GUI: Повідомлення
23	Видалити повідомлення з меню Повідомлень - повідомлення повинно зникнути		Перевірка GUI: Повідомлення

24	Натиснути на Будиночок в лівому повернемо кутку, користувач потрапляє на Головний екран		Перевірка GUI: Повідомлення
25	Створити 30 повідомлень, використовуючи стрілки, спуститися в самий низ - стрілки працюють, всі повідомлення показані		Перевірка GUI: Повідомлення
26	Створити 30 повідомлень, використовуючи стрілки, піднятися в самий верх - стрілки працюють, всі повідомлення показані		Перевірка GUI: Повідомлення
27	Відкрити другий замок, перевірити по логам, що для другого замку відсилається тільки dtmf		Другий замок
28	Включити функцію Другий замок в налаштуваннях GUI, зберегти - при дзвінку в GUI монітора відображена кнопка для відкриття другого замку і клавіатура для введення DTMF		Перевірка GUI: Другий замок

29	Вимкнути функцію Другий замок в налаштуваннях GUI, зберегти - при дзвінку в GUI монітора НЕ відображена кнопка для відкриття другого замку		Перевірка GUI: Другий замок
30	Перезавантажити пристрій, поточний стан роботи другого замку (вкл \ викл) коректно показано		Перевірка GUI: Другий замок
31	Натиснути на перемикач для другого замку - стан змінено		Перевірка GUI: Другий замок
32	При включенні функції, перемикач виділений синім кольором		Перевірка GUI: Другий замок
33	При виключенні функції, перемикач показаний неактивним (сірим кольором)		Перевірка GUI: Другий замок
34	Надіслати повідомлення, перевірити, що іконка для меню Повідомлення змінила свій колір на червоний		Перевірка GUI: Другий замок

35	Надіслати повідомлення, перевірити, що іконка для меню Повідомлення змінила свій колір на червоний, вийти з меню Повідомлення - іконка повернула синій колір		Перевірка GUI: Другий замок

36	Зробити GET-запит на отримання налаштувань DTMF-коду	**Очікуємо:**
		
		приклад:
		
		{
		
		"codes": {
		
		"first": {
		
		"use_default_value": **true**,
		
		"custom_value": "1"
		
		},
		
		"second": {
		
		"is_enabled": **true**,
		
		"value": "0"}
			Перевірка API: Другий замок

Продовження Додатку Б

36	Зробити GET-запит на отримання налаштувань DTMF-коду	} } Що важливо: Показані настройки двох параметрів - 1 і 2 замок Для замку 1: Показано, вибрано чи дефолтний значення (true) або кастомними (false), перевірити, чи сходиться інформація про це в веб-інтерфейсі Показано, яке кастомними значення вписано в поле для першого коду Для замку 2: Показано включена (false) або відключена (true) дана функція Показано, яке кастомними значення вписано в поле для першого коду	Перевірка API: Другий замок

37	Відправити запрос на зміну налаштувань: Замок 1 - включити дефолтне значення, вписати кастомне значення, Замок 2 включити, вписати кастомне значенне	*Приклад:*
		{
		"codes": {
		"first": {
		"use_default_value": **true**,
		"custom_value": "0"
		}
		"second":
		"is_enabled": **true**,
		"value": "*"
		}}}
		Очікуємо:
		200 OK, перевірити що дані застосували в веб-інтерфейсі і на пристрої (відкрити двері за допомогою замка 1, потім замку 2. Перевірити по логам, що для Замок 1 і 2 посилається DTMF при відкритті дверей
			Перевірка API: Другий замок

38	Відправити запрос на зміну налаштувань: Замок 1 - вписати кастомне значення, вимкнути дефолтне значення, Замок 2 - вписати кастомне значення, вимкнути другий замок	*Приклад:*
		
		{
		"codes": {
		
		"first": {
		
		"use_default_value": **false**,
		
		"custom_value": "0"
		},
		
		"second": {
		
		"is_enabled": **false**,
		
		"value": "*"
		}}}
		
		очікуємо:
		
		200 OK, перевірити що дані застосували в веб-інтерфейсі і на пристрої (відкрити двері за допомогою замка 1, замок 2 не повинен відображатися на моніторі при дзвінку)
		
		Перевірити по логам, що для Замок 1 надсилається XML, а не DTMF
			Перевірка API: Другий замок

39	Відправити запрос на зміну налаштувань: Замок 1 - вписати кастомне значення, вимкнути дефолтне значення, Замок 2 включити, вписати значення	*Приклад:*
		{
		"codes": {
		
		"first": {
		
		"use_default_value": **false**,
		
		"custom_value": "0"
		
		},
		
		"second": {
		
		"is_enabled": **true**,
		
		"value": "*"
		
		}
		}
		}
		Очікуємо:
		
		200 OK, перевірити що дані застосували в веб-інтерфейсі і на пристрої (відкрити двері за допомогою замка 1, потім замку 2. Перевірити по логам, що для Замок 1 надсилається XML, а не DTMF при відкритті дверей
			Перевірка API: Другий замок

40	Відправити запрос на зміну налаштувань: Замок 1 - вписати кастомне значення, включити дефолтне значення, Замок 2 вимкнути, вписати значення	*Приклад:*
		
		{
		"codes": {
		
		"first": {
		
		"use_default_value": **false**,
		
		"custom_value": "0"
		
		},
		
		"second": {
		
		"is_enabled": **false**,
		
		"value": "*"
		}}}
		очікуємо:
		
		200 OK, перевірити що дані застосували в веб-інтерфейсі і на пристрої (відкрити двері за допомогою замка 1, замок 2 не повинен відображатися на моніторі при дзвінку)
		
		Перевірити по логам, що для Замок 1 надсилається XML, а не DTMF
			Перевірка API: Другий замок

41	Відправити запит з порожнім параметром, що відповідає за значення DTMF для Замок 1,2 (negative)	*Приклад 1:*
		
		{
		
		"codes": {
		
		"first": {
		
		"use_default_value": **true**,
		
		"custom_value": ""
		
		},
		
		"second": {
		"is_enabled": **true**,
		"value": "*"
		}
		}
		}
		Приклад 2:
		
		{"codes": {
		"first": {
		"use_default_value": **true**,
		"custom_value": "0"
		},
		"second": {
		"is_enabled": **true**,
		"value": ""}}}
			Перевірка API: Другий замок

Продовження Додатку Б

42	Відправити запит з літерним параметром, що відповідає за значення DTMF для Замок 1,2 (negative)	Приклад:
		
		{
		
		"codes": {
		
		"first": {
		
		"use_default_value": **true**,
		
		"custom_value": "ghbdtn"
		
		},
		
		"second": {
		
		"is_enabled": **true**,
		
		"value": "0"
		
		}}}
			Перевірка API: Другий замок

43	Відправити запит з параметром, що відповідає за використання кастомного значення для Замок 1, що відрізняється від специфікації (negative)	*Приклад:*
		
		{
		
		"codes": {
		
		"first": {
		
		"use_default_value": use_default_value,
		
		"custom_value": "1"
		
		},
		
		"second": {
		
		"is_enabled": **true**,
		
		"value": "0"
		
		}
		
		}
		
		}
			Перевірка API: Другий замок

Продовження Додатку Б

44	Відправити запит з порожнім параметром, що відповідає за використання кастомного значення для Замок 2 (negative)	*Приклад:*
		
		{
		"codes": {
		"first": {
		"use_default_value": true,
		"custom_value": "1"
		},
		"second": {
		"is_enabled": **true**,
		"value": ""
		}
		}
		}
			Перевірка API: Другий замок

45	Відправити запит з параметром, що відповідає за використання значення для Замок 2 і відрізняється від специфікації (negative)	*Приклад:*
		
		{
		
		"codes": {
		
		"first": {
		
		"use_default_value": true,
		"custom_value": "1"
		},
		"second": {
		"is_enabled": **enable**,
		"value": "0"
		}
		}}
			Перевірка API: Другий замок

Продовження Додатку Б

46	Налаштувати на іншому моніторі однакову логічний адресу з тестованим пристроєм, поставити наприклад тестуємому монітору порядковий номер 0, другому монітору 1 і ввести різні коди синхронізації, відкрити список синхронізованих моніторів в меню Викликів - пристрої не показані		Синхронізація моніторів
47	Налаштувати на трьох інших моніторах однакову логічну адресу з тестованим пристроєм, поставити тестуємому монітору порядковий номер 0, другому монітору 1, третьому 2, четвертому 3 і однаковий код синхронізації (напр. 123456)		Синхронізація моніторів
48	Налаштувати на моніторі унікальну логічну адресу в мережі, перевірити список синхронізованих пристроїв - список порожній		Синхронізація моніторів

49	Відкрити список синхронізованих моніторів в меню Викликів - всі монітори показано відповідно до умови		Синхронізація моніторів
50	Перевірити, що в списку відображаються пристрої на платформах Android 4 \ 6, Camdroid, як Пристрій 0X (де X - порядковий номер)		Синхронізація моніторів
52	Перевірити на платформах Android 4 \ 6, Camdroid моніторах, що при дзвінку з \ на тестований пристрій, відображення відео з камери		Синхронізація моніторів
53	Зателефонувати з монітора, перевірити відображення іншого пристрою на моніторі - показано як Пристрій 0X (де X порядковий номер)		Синхронізація моніторів
54	Зателефонувати на монітор, перевірити відображення іншого пристрою на моніторі - в полі показано "Пристрій 0X" (де X - порядковий номер)		Синхронізація моніторів

55	Зателефонувати з монітора, перевірити відображення іншого пристрою на моніторі - показано логічну адресу (negative)		Синхронізація моніторів
56	Вибрати синхронізований пристрій, натиснути на "Зателефонувати на синхронізоване пристрій" - показано повідомлення "Дзвоню на Пристрій 0X" (де X - порядковий номер)		Синхронізація моніторів

57	Створити 30 повідомлень, зайти в в Повідомлення, а потім перевірити, що прийшли всі 30 повідомлень	*Приклад:*
		{"codes": {
		"first": {
		"use_default_value": use_default_value,
		"custom_value": "1"},
		"second": {
		"is_enabled": **true**,
		"value": "0"}}}
			Перевірка API: Повідомлення

58	Створити 30 повідомлень, перевірити, що повідомлення можна перегорнути за опомогою кнопки Вправа, щоб побачити інші	*Приклад:*
		{"codes": {
		"first": {
		"use_default_value": true,
		"custom_value": "1"},
		"second": {
		"is_enabled": **true**,
		"value": ""}}}
			Перевірка API: Повідомлення